SANS Security Essentials

GSEC Practical Assignment

Version 1.4b

Option 1

3 September 2003

Examining the RPC DCOM Vulnerability: Developing a Vulnerability-Exploit Cycle

By

Kevin O’Shea

~ Contents ~

Abstract

This paper proposes to build on the vulnerability life-cycle work first proposed by Arbaugh, Fithen and McHugh[1] to establish a detailed framework for vulnerability analysis. These extensions to the life-cycle, now proposed as the Vulnerability-exploit cycle, contain additional developmental stages intended to reflect recent experiences when analyzing critical events. In particular, The Remote Proceedure Call (RPC) Distributed Distributed Component Object Model (DCOM) buffer overrun vulnerability found in a multitude of Windows operating systems and Cisco devices / control programs is then deconstructed and charted against this revised vulnerability-exploit cycle. Further, the use of human intelligence, gathered through numerous security, hacker and cracker related websites, weblogs, user-groups, and discussion boards, will be shown to be a useful tool in capturing and documenting the evolution of the vulnerability. By developing a detailed framework in which to analyze events and milestones within the vulnerability-exploit cycle, critical events and time correlations can be recognized. This will lead to the ability to predict vulnerability and exploit behavior more effectively.

Introduction

At the very basic level, computers are simple machines: Digital data is stored through the manipulation of switches – which are either on or off. Process the data through an operation and get a result. The fundamental elements of computing technology have not changed since ENIAC was dedicated at UPENN in 1946[2]. Managing billions of switches and thousands of operations per second in a user friendly environment is where things get complicated. Over the past two decades, advances in user-friendly operating systems have allowed the power of the computer to be harnessed by untold millions of people. Features like plug and play, reduction in the need for command line driven functions, memory management, graphical user interfaces (GUI), etc. have made the computing experience more accessible to the general population. However, these same advances in ease of operation came with a cost manifested in the complexity of the operating system code. It is highly likely that errors will exist within the millions of lines of code[3] that comprise any operating system or complex program. Whether it be for the advancement of knowledge and malicious intent, exploiting programming flaws and errors has grown, evolved and flourished with the increasing complexity of programs and operating systems. The unprecedented growth of the Internet has seen an ever increasing number of systems connected to one another and to the world.

This increasing connectivity has created a situation where coding flaws in operating systems and other programs can be now be exploited remotely to yield elevated privileges to unauthorized users. A vulnerability can be exploited through direct attack via a scripted exploit or by a specially crafted virus or worm. It should be noted, however, that not all viruses or worms take advantage of a flaw in the software code to cause damage or to propagate themselves; rather some will rely on some assistance from the computer operator in order to be successful. For the purposes of this paper, viruses and worms will be discussed in the context of those that rely on flaws in the software code, rather than those that require operator assistance.

The flaws in software code affect all users; from entire economies of the most powerful nations in the world to the home user. It is estimated that the Slammer worm caused approximately $1 Billion in lost productivity worldwide in its first five days, but it barely broke the top-ten list for most damaging malware.[4] It is crucial that the events leading up to the release of a damaging piece of malware be critically examined. Determining how the underlying vulnerability gets discovered, released, patched, exploited, etc., is essential in learning the timeline of events associated with malware as a whole. Improving the ability to predict the development and spread of a new virus or worm can potentially save billions of dollars in lost productivity.

The Vulnerability-Exploit Cycle

The framework of a life-cycle model for vulnerabilities was proposed by Arbaugh, Fithen, and McHugh in an article titled "Windows of Vulnerability: A Case Study Analysis" in the December 2000 issue of the IEEE[5]. This paper proposed a model where a vulnerability would pass through multiple stages from Birth to Death. The stages they detailed included the following:

1. Vulnerability birth
2. Vulnerability discovery
3. Vulnerability disclosure
4. Correction / Release of a fix
5. Publication / Publicity Regarding Vulnerability
6. Scripting
7. Exploit Death

Using their framework, an expanded vulnerability-exploit cycle is proposed where additional stages have been added to allow for a greater degree of clarity when analyzing the event-timeline correlation associated critical milestones. These steps include the change of step five above from "Publicity" regarding the vulnerability to "Exploit Publication," and the addition of four new event stages.

The new vulnerability-exploit cycle, as proposed, would have the following steps:

1. Vulnerability Birth
2. Vulnerability Discovery
3. Vulnerability Disclosure
4. Vulnerability Correction / Fix Release
5. Exploit Creation and Publication
6. Manual Exploit Use in the Wild
7. Exploit Scripting and Automation
8. Automated Propagation Mitigation
9. Exploit Death

A detailed discussion of the revised vulnerability-exploit cycle follows:

1. Vulnerability Birth – The time in which the flaw or vulnerability is created in the software. This may occur at the time of initial coding, or may be the results of changes made to the software after updates, patches, or other software programs are added to a system.

2. Vulnerability Discovery – Discovery occurs when there is conscious acknowledgement of the vulnerability and, at least partially, its implications. It should not be considered discovery when a bug or flaw is encountered by an unknowing user. There are several individuals, companies, and research groups which specialize in discovering vulnerabilities, and have made this effort their primary focus. Often it is difficult to determine the motive(s) that drives the work to uncover these flaws.

3. Vulnerability Disclosure – Disclosure can be described as when the vulnerability is made public, usually by the person, persons, company, or research group that made the initial discovery. This can be achieved through a number of means including through the media, presentation at a conference, or posting the description of the vulnerability on a personal or business website. The most commonly encountered means is where the vulnerability is first posted on the research group's website and notices are forwarded to a security-related website for additional publicity.

4. Vulnerability Correction / Fix Release – Not long after a vulnerability is made public through disclosure, the manufacturer of the product will release a patch to fix the flaw in the program that created the vulnerability. Due to the complexity of current operating systems and other software, it is quite possible that a patch cannot be created or that the patch will affect other protocols, programs and interfaces; possibly even create another vulnerability on the target system. Instability following patching is a top reason for system administrators to resist applying the recommended patches and upgrades, and as a result, patches are often

not applied.[6]

Additional information on how to limit exposure to a vulnerability begins to enter into the public forum at this time. Often there will be defense techniques that when applied, either alone or in conjunction with the manufacturer's patch, will limit the ability of the vulnerability to affect a particular system or network. At this point in the vulnerability-exploit cycle, affected users are taking action to fix the underlying vulnerability, rather than mitigative actions against a virus or worm.

5. Exploit Creation and Publication – This stage was originally labeled the "Publicity" stage by Arbaugh, Fithen and McHugh in their vulnerability life-cycle. According to Arbaugh, et.al., the Publicity stage is defined as "…the vulnerability becomes known on a large scale once the disclosure gets out of control." This step appears to be redundant with their Disclosure step which they define as "…when the discoverer reveals details of the problem to a wider audience." Therefore, this step has been changed to "Exploit Publication" to define the event when a proof of concept code is publicly released.

   Depending on the independent disclosure policies of vulnerability researchers, an exploit against a vulnerability could be created, but never released. This also means that the creation of an exploit might occur minutes, days, weeks or months before it is published, and it may not be published by those that initially find the vulnerability. It is common for the initial rudimentary proof of concept exploit code to be put forward by named and attributable sources. Very often the exploit will be published in its raw code format and will require compilation before being able to be executed. The publication of exploit code occurs on the same channels and public forums where vulnerability disclosure takes place, Advancing the exploit into batch files, scripts, or GUIs is discussed below in Manual Exploit Use in the Wild and the crafting of a virus or worm based on the vulnerability is covered below in Exploit Automation / Scripting. A vulnerability may live out its entire life without an exploit being specifically crafted for it; many factors will come into play on this matter, but often the extent of use of the target software or device will drive the exploit creation and its longevity. Several variants of the original exploit will often be developed, as will completely new exploits that may use a different logical structure.

6. Manual Exploit Use in the Wild – In this stage of the vulnerability-exploit cycle, each new infection will occur as a direct result of manual human effort. This stage is first characterized by compiling the proof of concept exploit, and variants thereof, into an executable script with appropriate documentation. When an exploit script is written against a critical vulnerability, its use will often give an intruder elevated privileges and a

direct connection to the victim's system. The next logical step is for GUIs to be created for the scripted exploit to further increase its ease of use. The script/GUI may be partially automated or have automated components, i.e. a built in IP/port scanner and ftp client, but the process requires direct human involvement to compromise machines. Although the threat to the Internet as a whole is far less, the potential impact of unauthorized remote access to a particular system is far more dangerous.

7. Exploit Scripting and Automation – This stage in the vulnerability-exploit cycle is when an exploit is wrapped into some manner of propagation-enhancing code to be labeled a virus or worm. Not all vulnerabilities will lend themselves to conversion into a worm or a virus, and not all viruses or worms are based on exploiting a software flaw. This particular stage should be viewed as a potential milestone, rather than a certainty.

   Fully automating the exploit into a self-propagating worm involves the addition of code for scanning for vulnerable systems and a manner of transport to the infected systems. CERT wrote in their Overview of Attack Trends[7] that attack tools have become increasingly automated, commonly using advanced victim scanning patterns, scanning-compromise coupling and the ability to self propagate without human interaction. The primary "damage" caused by worms in recent history has been slowed and/or stopped Internet traffic as a result of their scanning activities. To date, most prolific worms have not carried an overtly malicious payload; however, the CERT Trends paper would point to the conclusion that a prolific worm with a malicious payload is a certainty in the future.

8. Automated Propagation Mitigation – This stage is characterized by the realization in the greater computer security community that the exploit has been automated into a worm or virus. This step is inexorably linked to the 'vulnerability fix/patch release' step, with a greater focus on mitigating the effects of the worm and a diminished focus on correcting the underlying vulnerability.

9. Exploit Death – At some point in time, the program for which the vulnerability exists will no longer be in use. This is realistically the only time when the exploit is truly dead. Although the popularity of a particular exploit may die off over time, and worm propagation may lessen as people either patch their systems or change their system configurations to limit the spread of the worm, the exploit will never truly be dead as long as systems in use remain vulnerable.

It is conceivable that the vulnerability may become stalled, perhaps permanently, at any of the above stages. For example, if a vulnerability in OS/2 was discovered and disclosed today, it is probable that an exploit would not be crafted for it because the number of affected systems is low and diminishing. Further, it is possible that several iterations for certain a steps may occur and multiple branches from the original iteration may develop. This is particularly true for steps four through eight, as new exploits are created for a particular vulnerability and scripted or wrapped in scanning and propagation code and launched into the wild.

The importance of the scripting and GUI program creation for an exploit cannot be overlooked. Many people rely on anti-virus software to protect them against a worm; but the worm is simply exploiting a vulnerability. If the underlying vulnerability is not fixed through a patch, the anti-virus software will have no effect against a direct attack using the published exploit. The scanning algorithms, propagation techniques, and payload of a worm can be dissected and analyzed as it is a static piece of code and will behave predictably. The damage that can be caused by an unauthorized user with elevated privileges can be devastating and far beyond the scope of the predictable damage caused by a worm.

Vulnerability-Exploit Cycle of the RPC DCOM Buffer Overrun Vulnerability

About the RPC DCOM Function

In order to learn more about the impact of the vulnerability, it is important to learn more about the protocol(s) in which the vulnerability exists. The Remote Procedure Call (RPC) is a protocol used by the Windows operating system to provide seamless inter-process communication between programs running on a local machine and a remote machine. The Distributed Component Object Model (DCOM) is a protocol that enables software components to communicate directly across multiple network transports, including Internet protocols such as HTTP.[8]

The RPC and DCOM protocols are linked in that the DCOM protocol listens for DCOM object activation requests that are sent by client machines on RPC enabled ports. The flaw exists in the DCOM interface and how malformed messages are handled and is not a flaw in the RPC process; however, it is through the RPC local/remote machine relationship that the malformed message can be passed along to the DCOM interface. This particular vulnerability, if exploited, would provide the intruder with full Local System privileges, where they would be allowed to create new accounts with privileges, install new programs, and view, modify, or delete data. To exploit this vulnerability, a specially crafted request would need to be sent to port 135, 139, 445, 539, or any other port configured for RPC functionality.[9] The malformed message delivered to the DCOM interface induces a buffer overrun error[10] which causes the machine to fail in such a way that arbitrary code can be executed remotely.[11]

Vulnerability Birth

The DCOM functionality began to make its way into the Windows operating system in the late year releases of Windows 95 in 1996. Macintosh and UNIX versions of DCOM became available in early 1997.[12] The birth of this vulnerability apparently occurred during the software development of this feature of the Windows OS, but it is possible that this flaw was part of an upgrade, patch, or update instituted from 1996 to present.

Vulnerability Discovery and Disclosure

On July 16, 2003, a research group called the “Last Stage of Delirium” updated a special section of their website to read “LSD Finds a Remote Vulnerability in Windows NT/2000/XP/2003 server.” Additional details on the vulnerability included the following description:

> This is a stack buffer overflow vulnerability that exists in an integral component of any modern Windows operating system, an RPC interface implementing Distributed Component Object Model services (DCOM). In a result of implementation error in a function responsible for instantiation of DCOM objects, remote attackers can obtain remote access to vulnerable systems.
>
> The impact of this vulnerability should be considered as critical. Throughout its exploitation, any user can gain complete control over a vulnerable system by the means of a remote attack. By sending specially crafted message to the TCP port 135 of vulnerable Windows system, an attacker can exploit the vulnerability and execute any code with SYSTEM privileges.
>
> The impact of the vulnerability can be hardly overestimated. It affects every installation of the Windows NT/2000/XP/2003 operating system not protected by additional security mechanisms for access control, such as firewall systems. The vulnerability may also cause enormous harm if its exploitation would be conducted with the usage of even primitive worm technologies (http://www.lsd-pl.net/special.html).[13]

It was clear through the mention of “any modern Windows operating system”, “remote access” and “complete control” that this vulnerability would have global implications.

The announcement on the lsd-pl.net website made mention of their work in successfully exploiting the RPC DCOM vulnerability in Windows 2000 (service packs 1-4), Windows XP (service pack 1) and Windows 2003 Server (regardless of the service packs installed). What was not clear in their initial disclosure was exactly how pervasive this vulnerability would be throughout the multitude of versions and service packs of the Windows operating system, including other protocols and programs that use the RPC DCOM function in some way.

Following the initial discovery and disclosure statements from LSD, the list of vulnerable systems and devices would continue to grow over the next few days and weeks. Bugtraq at www.SecurityFocus.com now lists no less than 109 separate programs (last validated 18 August 2003), devices, or operating systems which are susceptible to the RPC DCOM buffer overrun vulnerability. Table 1 shows the full list from Security Focus:

Table 1 – List of Programs/Devices/Operating systems Vulnerable to the RPC DCOM Buffer Overrun – List as of 18 August 2003 – Source Security Focus.com <http://www.securityfocus.com/bid/8205>

Cisco Access Control Server
Cisco Broadband Troubleshooter
Cisco CiscoWorks VPN/Security Management Solution
Cisco Collaboration Server
Cisco DOCSIS CPE Configurator
Cisco Intelligent Contact Management
Cisco Internet Service Node
Cisco IP Telephony Environment Monitor
Cisco Lan Management Solution
Cisco Media Blender
Cisco Networking Services for Active Directory
Cisco QoS Policy Manager
Cisco Routed Wan Management
Cisco Secure Policy Manager 3.0.1
Cisco Secure Scanner
Cisco Service Management
Cisco Small Network Management Solution
Cisco SN 5420 Storage Router 1.1 (7)
Cisco SN 5420 Storage Router 1.1 (5)
Cisco SN 5420 Storage Router 1.1 (4)
Cisco SN 5420 Storage Router 1.1 (3)
Cisco SN 5420 Storage Router 1.1 (2)
  - Microsoft Windows 2000 Workstation
  - Microsoft Windows 2000 Workstation SP1
  - Microsoft Windows 2000 Workstation SP2
  - Microsoft Windows 95
  - Microsoft Windows 98
  - Microsoft Windows ME
  - Microsoft Windows NT 4.0
  - Microsoft Windows NT 4.0 SP2
  - Microsoft Windows NT 4.0 SP3
  - Microsoft Windows NT 4.0 SP4
  - Microsoft Windows NT 4.0 SP5
  - Microsoft Windows NT 4.0 SP6
  - Microsoft Windows NT 4.0 SP6a
Cisco SN 5420 Storage Router 1.1.3
Cisco Trailhead
Cisco Transport Manager
Cisco Unity Server
Cisco Unity Server 2.0
Cisco Unity Server 2.1
Cisco Unity Server 2.2
Cisco Unity Server 2.3
Cisco Unity Server 2.4
Cisco Unity Server 2.46
Cisco Unity Server 3.0
Cisco Unity Server 3.1
Cisco Unity Server 3.2
Cisco Unity Server 3.3
Cisco Unity Server 4.0
Cisco uOne 1.0
Cisco uOne 2.0
Cisco uOne 3.0
Cisco uOne 4.0
Cisco User Registration Tool
Cisco Voice Manager

Microsoft Windows 2000 Advanced Server SP2
Microsoft Windows 2000 Advanced Server SP1
Microsoft Windows 2000 Advanced Server
Microsoft Windows 2000 Datacenter Server SP4
Microsoft Windows 2000 Datacenter Server SP3
Microsoft Windows 2000 Datacenter Server SP2
Microsoft Windows 2000 Datacenter Server SP1
Microsoft Windows 2000 Datacenter Server
Microsoft Windows 2000 Professional SP4
Microsoft Windows 2000 Professional SP3
Microsoft Windows 2000 Professional SP2
Microsoft Windows 2000 Professional SP1
Microsoft Windows 2000 Professional
Microsoft Windows 2000 Server SP4
Microsoft Windows 2000 Server SP3
Microsoft Windows 2000 Server SP2
Microsoft Windows 2000 Server SP1
Microsoft Windows 2000 Server
Microsoft Windows NT Enterprise Server 4.0 SP6a
Microsoft Windows NT Enterprise Server 4.0 SP6
Microsoft Windows NT Enterprise Server 4.0 SP5
Microsoft Windows NT Enterprise Server 4.0 SP4
Microsoft Windows NT Enterprise Server 4.0 SP3
Microsoft Windows NT Enterprise Server 4.0 SP2
Microsoft Windows NT Enterprise Server 4.0 SP1
Microsoft Windows NT Enterprise Server 4.0
Microsoft Windows NT Server 4.0 SP6a
Microsoft Windows NT Server 4.0 SP6
Microsoft Windows NT Server 4.0 SP5
Microsoft Windows NT Server 4.0 SP4
Microsoft Windows NT Server 4.0 SP3
Microsoft Windows NT Server 4.0 SP2
Microsoft Windows NT Server 4.0 SP1
Microsoft Windows NT Server 4.0
Microsoft Windows NT Terminal Server 4.0 SP6a
Microsoft Windows NT Terminal Server 4.0 SP6
Microsoft Windows NT Terminal Server 4.0 SP5
Microsoft Windows NT Terminal Server 4.0 SP4
Microsoft Windows NT Terminal Server 4.0 SP3
Microsoft Windows NT Terminal Server 4.0 SP2
Microsoft Windows NT Terminal Server 4.0 SP1
Microsoft Windows NT Terminal Server 4.0
Microsoft Windows NT Workstation 4.0 SP6a
Microsoft Windows NT Workstation 4.0 SP6
Microsoft Windows NT Workstation 4.0 SP5
Microsoft Windows NT Workstation 4.0 SP4
Microsoft Windows NT Workstation 4.0 SP3
Microsoft Windows NT Workstation 4.0 SP2
Microsoft Windows NT Workstation 4.0 SP1
Microsoft Windows NT Workstation 4.0
Microsoft Windows Server 2003 Datacenter Edition
Microsoft Windows Server 2003 Datacenter Edition 64-bit
Microsoft Windows Server 2003 Enterprise Edition
Microsoft Windows Server 2003 Enterprise Edition 64-bit
Microsoft Windows Server 2003 Standard Edition

The researchers at LSD did not publish their proof of concept exploit code initially, but chose to work with Microsoft toward the development of a patch for the newly discovered vulnerability. This may place the true 'Discovery' date a few days prior to the published disclosure date, as their published statement leads one to believe that contact with Microsoft had occurred prior to their announcement.

Vulnerability Correction / Fix Release

Microsoft was quick to release a patch for the vulnerable systems as the Microsoft Security Bulletin MS03-026 was released on July 16, 2003. MS03-026 was titled "Buffer Overrun In RPC Interface Could Allow Code Execution (823980)" and detailed the vulnerability and affected systems. The Security Bulletin was continually updated to include new versions of the Mircosoft Windows operating system, with the last revision dated August 25, 2003.[14] It is apparent from the timeliness of the release of the Security Bulletin and the completeness of the information contained therein, that Microsoft took this vulnerability seriously.

Exploit Creation and Publication

It was implied in LSD's disclosure that they had in fact created proof of concept exploit code prior to their public disclosure of the exploit, but it was never released to the public. On July 21, 2003, Benjurry of the X-Focus Research Team posted a rudimentary proof of concept on the Full Disclosure mailing list written by his team member, Flashsky.[15] This would be refined and officially posted on the X-Focus Team website on July 25, 2003. The posting stated that they had in fact developed proof of concept code to exploit the RPC DCOM buffer overrun vulnerability.[16] Interestingly, in light of the extensive list of affected programs maintained by Bugtraq, the initial proof of concept listed only Windows 2000 Service Pack 2 and 3 as potentially vulnerable systems. The X-Focus Team's proof of concept code was followed closely by the publication of *dcom.c* by H.D. Moore (HDM) of The Metasploit Project[17]. The first notice of disclosure of the *dcom.c* exploit on the Security Focus forums occurred on July 26, 2003 at 5:25PM and was posted by a user identifying themselves/their email as "fulldisclosure catholic org". Catholic.org is a religious portal with an anonymous email system. The Metasploit project also hosted the *dcom.c* exploit, but a time and date stamp was not provided on the Metasploit page. In comments regarding the X-Focus exploit code, HDM stated that he had improved upon Flashsky and Benjurry's previous exploit because "I don't like broken exploits, so I fixed it." The news of HDM's improved exploit code spread quickly through the security, hacker, and cracker communities, generating endless threads on discussion boards regarding compiling the code, which offsets are required for which systems, etc. Tracking the *dcom.c* exploit was made considerably easier by text that was inserted into the body of the exploit. The words “MARB”, “MEOW”, “MEOW”, and “MEOW” are easily noticed in an ASCII view of the exploit code and are present as strings in the executable. I believe it is necessary to include HDM’s *dcom.c* in the exploit publication phase, as it appears that many successive exploits, and the MSBlast worm, all used the *dcom.c* exploit as their exploit code foundation. It is unclear how the RPC DCOM vulnerability would have evolved differently if HDM had not clarified the X-Focus Team’s code.

Manual Exploit Use in the Wild

Following the publication of HDM’s *dcom.c*, the code was modified and used as the foundation for other scripted exploits. The changes and additions that were made to the exploit affected the ease of use of the exploit and increased the number of systems against which the exploit would be effective. The four keywords of “MARB” and “MEOW” are present in the raw code in all but one of the below exploits; the exploit code for *KaHT II* does not have the telltale keywords of the initial *dcom.c*. This is an important to note because this would mean that there was little, if any, concurrent development of a different exploit or that concurrent development halts once a working exploit is released.

The table below shows the release date, name of the exploit and the author of the variants of the HDM exploit as published by Security Focus.[18] Obtaining exact release dates for the new exploits can be problematic, unless a release date is noted in the code itself and/or is integrated into the filename. Estimated release dates have been provided for four of the eight variants based on anecdotal evidence.

Table 2 – Variants of the HDM *dcom.c* exploit

DATE	Exploit Name	Author – Website
7-29-2003	07.29.dcom18.c	K-Otic; //k-otic.com
7-30-2003	07.30.dcom48.c	K-Otic; //k-otic.com
7-30-2003	30.07.03.dcom.c	H.D. Moore; www.metasploit.com
7-30-2003*	Dcomrpc.c	Not annotated re: author
7-30-2003*	Dcom_Expl_UnixWin32	Benjamin Lauzière; altern.org
8-4-2003	0x82-dcomrpc_usemgret.c	you dong-hun, //x82.i21c.net

8-4-2003*	oc192-dcom.c	//oc192.us
8-4-2003*	KaHT II	//3wdesign.es

*Estimated

Soon after the flurry of releases of modified and functionally expanded exploits, a graphical user interface (GUI) was published to increase the ease of exploiting computers with the RPC DCOM vulnerability. On August 11, 2003, the Astalavista Group[19] updated the 'Tools' section of their homepage to include a program called *RPC GUI* created by r3L4x of the DSK Coding Crew[20]. Although the release date of August 11th was noted by Astalavista, all the files in the downloadable zipped archive were dated August 7th. This date distinction becomes very important when viewed against the release of the MSBlast worm, detailed below. It is unclear as to which exploit was used as the basis for the *RPC GUI*, but it is apparent it was the *dcom.c* or a direct variant thereof.

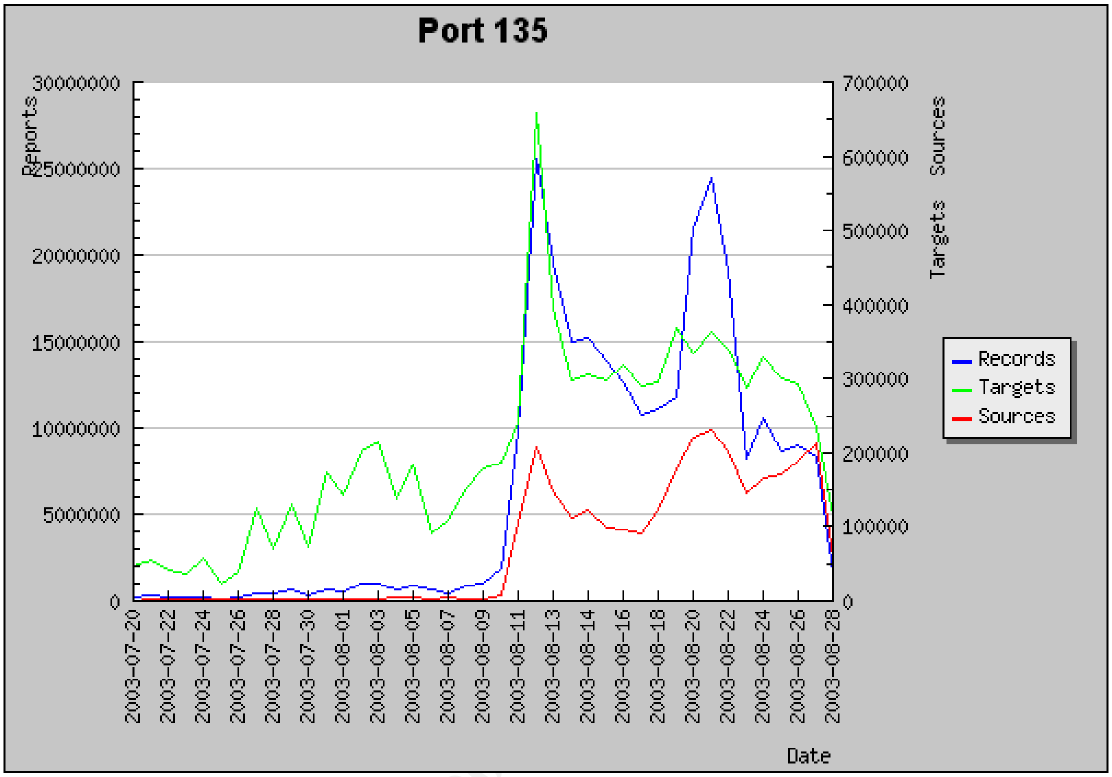
Improvements were soon made to the *RPC GUI,* and *RPC GUI v2* was released to Astalavista on August 13, 2003. As with the initial program, this program was created by r3L4x of the DSK Coding Crew. On the Astalavista's description of the program, it was noted that the GUI was created by r3L4x and the exploit was created by oc192 Security. The exploit created by oc192 Security, *oc192-dcom.c* was much improved on the original *dcom.c* in that it simplified the process of guessing the targets specific version of their Windows OS and service pack down to two universal offsets. Additionally it did not crash the victim's computer when the attacker terminated the connection.[21]

Both of the programs detailed above are very simple and very powerful and contain both a built in port scanner and an ftp client. These programs combine all the required tools to target, compromise, and upload content to victims' systems.

Exploit Scripting and Automation

Since the release of the RPC DCOM vulnerability on July 16th, the Internet security community waited guardedly for the release of a worm that would take advantage of this sweeping flaw in the world's most widely used computer operating system. Late in the day (EDT) on August 11, 2003, those monitoring system-wide Internet traffic began to see a marked increase in scanning traffic to port 135, the main port of entry to exploit the RPC DCOM vulnerability.[22] The graph of scanning activity as recorded by the Internet Storm Center, operated by SANS, shows this marked increase in the number of scanning sources, targets and records on August 11, 2003 on port 135 (Chart 1).

Chart 1 – Scanning Activity on Port 135. Courtesy of SANS.
Notice marked increase in activity corresponding to the release of the initial MSBlast worm on August 11, 2003.



Verification from a number of sources including CERT, Sophos, Symantec, McAfee and ISS soon followed indicating that in fact the worm based on the RPC DCOM vulnerability had been released. The binary of the executable was named MSBlast.exe, leading to its common name, the MSBlast worm. Several aliases were also noted; MSBlast, Lovsan, W32.Blaster.worm, W32/Lovsan, and Win32.Posa.Worm.

The worm worked by using the RPC DCOM exploit through port 135 to gain elevated access where a remote shell on port 4444 would be established back to the source. The source would then execute a tftp command to pull the binary to the target.

The worm itself contained a number of interesting strings including:

I just want to say LOVE YOU SAN!!
billy gates why do you make this possible ? Stop making money and fix your software!!
MARB
MEOW
MEOW

MEOW

The first two strings in the code received the majority of the media coverage for its disparaging comments toward Microsoft and led to the Lovsan alias for the worm; however, the presence of the keywords found in the initial *dcom.c* exploit was more important in following the evolution of the vulnerability. It was abundantly clear that the creator of the worm appended scanning and propagation code to the *dcom.c* exploit or one of its direct descendants and that new exploit code was not written specifically for this worm. This was also an important because the *dcom.c* exploit had been published weeks earlier and its methods of operation had been carefully studied and analyzed, and therefore a large portion of how the worm operated was already well understood.

The MSBlast worm had a relatively temperate scanning algorithm, and therefore the propagation of this worm was far less rampant than other recent worms such as SQL Slammer or NIMDA. Although the scanning algorithm was not as aggressive as other recent worms, there were several accounts of infected persons not being able to download a patch for a rebuilt system before new infections established themselves.

The MSBlast worm had no self-contained payload; it relied on causing an infected machine to establish a tftp connection back to the source to upload the worm binary. Damage to the victim's machine is limited to an entry made in the registry and the usage of bandwidth during its scanning routine(s). The worm did contain instructions for the infected machine to launch a SYN Flood attack against the windows update website, www.windowsupdate.com, on every day from August 16th till the end of December and after the 15th of every successive month.[23] Notwithstanding the predictable behavior of the MSBlast worm, a remote connection to an unauthorized user is made to upload the binary, which caused significant concern in the security community.

Several new worms would be released over the following weeks, all of them functionally the same as the original MSBlast and differentiated only by its binary name and registry key changes. Below is a table showing the four main MSBlast worm variants as of August 22, 2003.

Table 3 – Four primary MSBlast worm variants.

Release Date	**Binary Name**	**Different Registry Key Entry?**
8-6-2003	MSPatch.exe	Yes
8-13-2003	teekids.exe	Yes
8-13-2003	penis32.exe	No

8-18-2003	dllhost.exe and svchost.exe (Welchia / Nachi*)	Yes

*Common name of worm variant.

The last worm variant noted above, the Welchia / Nachi worm, is the most interesting. This worm scans for vulnerable systems and uses exploits against the RPC DCOM vulnerability and a Web DAV vulnerability to gain remote "Local System" rights. The worm then deletes the MSBlast process and applies the Microsoft patch to remedy the RPC DCOM vulnerability. The worm is then programmed to delete itself when the system clock reaches Jan 1, 2004.[24] Although this worm had good intentions, the scanning technique used by the worm can generate a significant volume of ICMP Host-unreachable messages.[25] In some networks where the Welchia / Nachi worm became established, network traffic was brought to a halt due to the worm's scanning activities. This variant of the MSBlast worm was apparently the only one of the primary variants to cause at least localized network problems due to increased traffic issues. Affected organizations that admitted their systems were affected by the Welchia / Nachi worm included the US Navy[26] and Air Canada[27], who was forced to delay and cancel flights due to disruption in their reservation center's computers.

Automated-Propagation Mitigation

The security community responded by dissecting the operation of the MSBlast worm and its variants. Recommendations were made by the security community to block all incoming ports that would commonly carry the inbound worm, primarily TCP/UDP 135, 137, 139, and 445. Additional recommendations called for blocking of outbound port TCP/UDP 4444, the port on which a remote shell would be established.

Exploit Death

Based on the widespread and worldwide use of the Windows operating system, and the many versions and variants affected by the worm, it is clear that the underlying vulnerability will exist in the wild, un-patched, for the foreseeable future. The use of the exploit to gain remote access will fade as new exploits become available[28] and the continued propagation of the existing worm(s) will level off and decline over time; however, as long as vulnerable systems are connected to other computers through the Internet or LAN, this exploit will continue to live on.

Conclusion

The vulnerability-exploit cycle presents a comprehensive birth to death sequence as a framework to analyze critical vulnerabilities. By looking more in depth at these quasi-predictable stages, one can begin to uncover relationships between events and time sequences. By examining the “human element” found in discussion boards, groups, weblogs, and chat channels, additional details about the evolution of an exploit can be learned beyond the reactive and mitigative information often associated with patching vulnerabilities and repairing the damage caused by the scripted or automated exploits. It is the author’s goal to raise the awareness of the importance of gathering and analyzing human intelligence as part of predicting vulnerability exploitation.

Appendices

Appendix A - Research Methodology

This research project began days before the release of the LSD notification of the RPC DCOM exploit. It was the author's intent to analyze a historic vulnerability and document, where possible, the dates associated with milestones within the vulnerability-exploit cycle and the potential human intelligence related to these events. It was noted, however, that analysis of Internet published data would be difficult based on the sheer volume of information that any particular vulnerability/exploit/worm generates throughout its entire existence. The publishing of the LSD notice on the RPC DCOM exploit therefore came at an opportune time to provide a proof of concept regarding documenting the sources of information related to milestones in the vulnerability-exploit cycle.

The first step was to generate a map of those sites that had a published affiliation with the LSD research group. This was done through the use of Google's link analysis function[29] to learn of the pages that Google had indexed that linked back to www.lsd-pl.net. This was then repeated for all of the sites that linked to the LSD research group site. Even after only two iterations deep, patterns and circular references became apparent, which allowed conclusions to be made on which sites were highly regarded by the overall community. Usenet archives, available through Google Groups, were also searched using the LSD website name and other related search terms including "RPC", "DCOM", and website names discovered through the link analysis. It was through this methodology that the circle of active and relevant websites to monitor over the next few weeks was developed.

The table below shows an example of a link analysis result and how the links will change over time as significant events occur that involve the link target. The result of the link analysis for www.lsd-pl.net conducted July 22, 2003 and the additions as of August 26, 2003 is presented in the table. Inferences can be made as to the notoriety gained through the publication of a major vulnerability discovery thought the additional links from new sources.

Additional research is needed to further validate, expand, and automate the link analysis process for determining the sources of actionable human intelligence; but the author believes that the use of this methodology provided valuable insight into the interconnectivity of the security, hacker, and cracker communities.

Table 4: Example of the Link Analysis Methodology, used on the LSD research group website. Table also shows changes to “link from” over time.

“ * “ denotes sites that exist on both the July 22 and August 26 analysis.

Link Target: www.lsd-pl.net	
July 22, 2003	**August 26, 2003**
www.secmod.com/news/2003-03-10/	packetstorm.icx.fr/whatsnew20.html
www.staropenoffice.com*	www.staropenoffice.com/*
www.blackhat.com/html/bh-usa-01/ bh-usa-01- schedule.html *	www.securityfocus.com/news/6519
active-security.org/arc8-2000.html*	www.blackhat.com/html/bh-usa-01/ bh-usa-01-schedule.html *
www.tietokone.fi/pda/uutinen.asp?news_id=19107	planeta.terra.com.br/informatica/gleicon/links.html
sigurnost.linux.org.ba/*	www.microsoft.com/korea/technet/security/ bulletin/MS03-026.asp?bPrint=True
raptor.antifork.org/	komputery.wp.pl/
www.techiwarehouse.com/Security	bama.ua.edu/~crock/ua_unix/0083.html
www.linux-sec.net/Exploits/*	www.ciwunion.com/
z0mbie.host.sk*	www.metasploit.com/shellcode.html
www.void.ru/content/1056 *	webnews.html.it/news/sendnews.php?idnews=1077
pages.cpsc.ucalgary.ca/~arlt/security/hackers.html *	active-security.org/arc8-2000.html*
dcortesi.com/2003-01-02/492.html	www.0xdeadbeef.info/
farking.spunge.org/links.html	www.security.org.sg/webdocs/news/event13.html
www.bosen.blogspot.com/	212.100.234.54/content/55/31797.html
www.efn.org/~gchu/	www.cs.rochester.edu/~bukys/weblog/archives/security/
packetstormsecurity.org/0108-exploits/	sigurnost.linux.org.ba/*
www.safemode.org/links.html	security.opennet.ru/base/hp/1055263092_2258.txt. html
www.sgi.ethz.ch/secadv/msg00086.html *	www.underground.org.pl/gminick/
www.rstream.net	www.linux-sec.net/Exploits/ *
	z0mbie.host.sk/*
	news.com.com/2100-1002-991041.html
	archives.neohapsis.com/archives/hp/2003-q1/0028.html
	www.csm.ornl.gov/~dunigan/security.html
	www.blackhat.com/html/bh-asia-02/ bh-asia-02-speakers.html
	msgs.securepoint.com/cgi-bin/get/bugtraq0008/152.html
	pages.cpsc.ucalgary.ca/~arlt/security/hackers.html *
	www.thinkingit.net/

	ciac.llnl.gov/ciac/bulletins/k-066.shtml
	securecomputing.stanford.edu/alerts/sendmail-vuln.html
	www.hacking.pl/
	www.spirit.com/Network/net0800.html
	www.tno.nl/instit/fel/intern/wkisec15.html
	www.theregister.co.uk/content/55/31797.html
	www.hivercon.com/hc02/speaker-lsd.htm
	www.sgi.ethz.ch/secadv/msg00086.html *
	idea.sec.dico.unimi.it/~andrew/
	www.ccc.de/congress/2001/fahrplan/event/260.en.html
	zdnet.com.com/2100-1105-991041.html
	www.sign.net.pl/article.php?sid=1363
	xforce.iss.net/xforce/alerts/id/147
	www.ptnix.com/
	cgi4.zdnet.co.jp/g/01_0a03032270_/news/0303/06/ne00_sendmail.html
	it.monitor.hr/index.php
	dmoz.org/Computers/Security/Hackers/
	void.ru/ *
	www.internet-magazine.com/news/view.asp?id=3557
	www.scribbler.ca/

References

[1] Arbaugh, William, A; Fithen, William, L; McHugh, John. “Windows of Vulnerability: A Case Study Analysis”. IEEE. 2000. URL: http://www.cs.umd.edu/~waa/pubs/Windows_of_Vulnerability.pdf (12 July 2003).

[2] Weik, Martin H. “The ENIAC Story.” 1961. Ordnance Ballistic Research Laboratories, Aberdeen Proving Ground, MD. URL:http://ftp.arl.mil/~mike/comphist/eniac-story.html (10 August 2003).

[3] Shankland, Stephen. “Governments to see Windows code.” CNET News. 14 January 2003. URL:http://news.com.com/2100-1001_3-980666.html (10 August 2003).

[4] Lemos, Robert. “Calculating the cost of Slammer.” 3 February 2003. URL:http://zdnet.com.com/2100-1104-982955.html (25 August 2003).

[5] Arbaugh, William, A; Fithen, William, L; McHugh, John. “Windows of Vulnerability: A Case Study Analysis”. IEEE. 2000. URL:http://www.cs.umd.edu/~waa/pubs/Windows_of_Vulnerability.pdf (12 July 2003).

[6] Naraine, Ryan. “When Patches Aren’t Applied.” Internetnews.com. March 28, 2003. URL:http://www.internetnews.com/dev-news/article.php/2171781 (25 July 2003).

[7] CERT Coordination Center. “Overview of Attack Trends.” 2002. URL:http://www.cert.org/archive/pdf/attack_trends.pdf (5 August 2003).

[8] Microsoft Security Bulletin MS03-026. 16 July 2003. Last updated 25 August 2003. URL:http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS03-026.asp (Technical Information). (July and August 2003).

[9] Ibid.

[10] Donaldson, Mark. “Inside the Buffer Overflow Attack: Mechanism, Method and Prevention.” SANS Institute. 3 April 2003. URL:http://www.sans.org/rr/paper.php?id=386 (6 August 2003).

[11] Microsoft Security Bulletin MS03-026. 16 July 2003. Last updated 25 August 2003. URL:http://www.microsoft.com/technet/treeview/default.asp?url= /technet/security/bulletin/MS03-026.asp (Frequently Asked Questions). (July and August 2003).

[12] Microsoft Corporation. “DCOM Technical Overview.” November 1996.

URL:http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndcom/html/msdn_dcomtec.asp (July 16, 2003).

[13] LSD Research Group. “Buffer Overrun in Windows RPC Interface.” 16 July 2003. URL:http://www.lsd-pl.net/special.html (16 July 2003).

Microsoft Security Bulletin MS03-026. 16 July 2003. Last updated 25 August 2003. URL:http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS03-026.asp (Technical Information). (July and August 2003).

Benjurry, “Microsoft Windows 2000 RPC DCOM Interface DOS AND Privilege Escalation Vulnerability.” Full Disclosure Mailing List. Mon, 21 Jul 2003 23:53:03. http://lists.netsys.com/pipermail/full-disclosure/2003-July/006851.html (25 August 2003)

Flashsky, “Microsoft Windows 2000 RPC DCOM Interface DOS AND Privilege Escalation Vulnerability.” X-Focus. 25 July 2003. URL:http://www.xfocus.org/advisories/200307/4.html (26 July 2003).

Metasploit project homepage: URL:http://www.metasploit.com/index.html

[18] Security Focus. “Microsoft Windows DCOM RPC Interface Buffer Overrun Vulnerability (Exploit).” 2 and 11 August 2003. URL:http://www.securityfocus.com/bid/8205/exploit/ (12 August 2003).

[19] Astalavista Group homepage: URL:http://www.astalavista.com (25 August 2003)

[20] Dark Side of Kalez homepage: URL:http://dcc.darksideofkalez.com/index1.htm

[21] RPC Exploit GUI v2. Astalavista.com. 13 August 2003. URL:http://www.astalavista.com/tools/auditing/network/multiscanner/ (15 August 2003)

[22] SANS. Port 135 scanning graph. URL:http://isc.incidents.org/port_details.html?port=135 (25 August 2003).

[23] eEye Digital Security. “Blaster Worm Analysis.” 11 August 2003. URL:http://www.eeye.com/html/Research/Advisories/AL20030811.html (12 August 2003).

[24] Network Associates. “W32/Nachi.worm.” 18 August 2003. URL:http://vil.nai.com/vil/content/v_100559.htm (25 August 2003).

[25] True Secure. “TruSecure Alert - Welchia worm.” 21 August 2003.

URL:http://www.trusecure.com/knowledge/hypeorhot/2003/welchia.shtml (25 August 2003).

[26] Verton, Dan. “Update: Navy says intranet hit by worm but still functioning.” Computer World. 19 August 2003. URL:http://www.computerworld.com/securitytopics/security/ story/0,10801,84158,00.html (27 August 2003).

[27] Lemos, Robert. “Good worm, new bug mean double trouble.” ZD Net Austrailia. 20 August 2003. URL:http://www.zdnet.com.au/newstech/security/ story/0,2000048600,20277472,00.htm (27 August 2003).

[28] Eschelbeck, Gerhard. “Laws of Vulnerabilities.” Qualys. July 30, 2003. URL:http//www.qualys.com/news/pr/index.php?page=pr_07_30_03c&lk=hm_pg (6 August 2003).

[29] Google Help. URL:http://www.google.com/help/features.html#link (22 August 2003).