*Oriental Journal of Computer Science & Technology* Vol. **1**(2), 151-154 (2008)

# Blended attacks exploits, vulnerabilities and Buffer-overflow techniques in computer viruses

**A.VENKATESWARA RAO* and MANDAVA V. BASAVESWARA RAO[1]**

*Department of Computer Science, Noble Institute of Science & Technology, Visakhapatnam (India).
[1]Sadineni Chowdariah College of Arts & Science, Maddirala, Chilakaluripet (India).

(Received: October 25, 2008; Accepted: Decembe 07, 2008)

## ABSTRACT

Exploits, vulnerabilities, and buffer-overflow techniques have been used by malicious hackers and virus writers for a long time. However, until recently, these techniques were not commonplace in computer viruses. The CodeRed worm was a major shock to the antivirus industry since it was the first worm that spread not as a file, but solely in memory by utilizing a buffer overflow in Microsoft IIS. Many antivirus companies were unable to provide protection against CodeRed, while other companies with a wider focus on security were able to provide solutions to the relief of end users. Usually new techniques are picked up and used by copycat virus writers. Thus, many other similarly successful worms followed CodeRed, such as Nimda and Badtrans. In this paper, the authors will not only cover such techniques as buffer overflows and input validation exploits, but also how computer viruses are using them to their advantage. Finally, the authors will discuss tools, techniques and methods to prevent these blended threats.

**Key words:** Vulnerabilities and Buffer-overflow techniques, computer viruses.

## INTRODUCTION

### Definition of a Blended Attack

A blended threat is often referred as a "blended attack." Some people refer to it as combined attacks" or "mixed techniques." We are not attempting to make a strong definition for this term, but we wish our readers to understand that we use the term "blended attack" in the context of computer viruses where the virus exploits some sort of security flaw of a system or application in order to invade new systems. A blended threat exploits one or more vulnerabilities as the main vector of infection and may perform additional network attacks such as a denial of service against other systems.

Security exploits, commonly used by malicious hackers, are being combined with computer viruses resulting in a very complex attack, which in some cases goes beyond the general scope of antivirus software. In general, a large gap has existed between computer security companies, such as intrusion detection and firewall vendors and antivirus companies. For example, many past popular computer security conferences did not have any papers or presentations dealing with computer viruses. Thus, apparently some computer security people do not consider computer viruses seriously as part of security, or they ignore the relationship between computer security and computer viruses.

When the CodeRed worm appeared, there was obvious confusion about which genre of computer security vendors could prevent, detect, and stop the worm. Some antivirus researchers argued that there was nothing that they could do about CodeRed, while others tried to solve the problem with various sets of security techniques, software, and detection tools to support their customers' needs. Interestingly, such intermediate solutions were often criticized by antivirus researchers. Instead of realizing the affected customers' need for such tools, some antivirus researchers suggested that there was nothing to do but to install the security patch. Obviously, this step is very important in securing the systems.

However, the installation of a patch on thousands of systems may not be easy to deliver at large corporations. Furthermore, corporations may have the valid fear that new patches could introduce a new set of problems, compromising system stability. CodeRed (and blended attacks in general) is a problem that needs to be taken care of by antivirus vendors as well as by other security product vendors, so that multi-layered security solutions can be delivered in a combined effort to deal with blended attacks.

## Background

The origin of blended attacks begins in November 1988, which was the year that the Morris worm was introduced. The Morris worm exploited flaws in standard applications of BSD systems:

It tried to utilize remote shell commands to attack new machines by using rsh from various directories. It demonstrated the possibility of cracking password files. The worm attempted to crack passwords in order to get into new systems. This attack was feasible because the password file was accessible and readable by everyone. Although the password file was encrypted, someone could attempt to encrypt test passwords and then compare them against the encrypted ones. The worm used a small dictionary of passwords that the author of the worm believed to be common or weak. Looking at the list of passwords in the author's dictionary, we have the impression that this was not the most successful of the worm's attacks.

If the previous step failed, the worm attempted to use a buffer-overflow attack against VAX-based systems running a vulnerable version of fingerd. (More details on this attack are available later on). This resulted in the automatic execution of the worm on a remote VAX system. The worm was able to execute this attack from either VAX or Sun® systems, but the attack was only successful against targeted VAX systems. The code was not in place to identify the remote OS version and thus, the same attack was used against the fingerd program of Sun's running BSD. This resulted in a core dump (crash) of fingerd on targeted Sun systems.

The Morris worm also utilized the DEBUG command of the sendmail application. This command was only available in early implementations of sendmail. The DEBUG command made it possible to execute commands on a remote system by sending an Simple Mail Transfer Protocol (SMTP) message. This command was a potential mistake in functionality and was removed in later versions of sendmail. When the DEBUG command was sent to sendmail, someone could execute commands as the recipient of the message.

Nevertheless, the worm was not without bugs. Although the worm was not deliberately destructive, it overloaded and slowed down machines so much that it was very noticeable after repeated infections occurred. Thirteen years later, in July, 2001, CodeRed repeated a very similar set of attacks against vulnerable versions of Microsoft Internet Information Server (IIS) systems. 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. Further information on the buffer-overflow attacks is made available in this paper (without any working attack code).

## Types of Vulnerability

### Buffer overflows

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.

For example, imagine an empty 33 cl. glass. This is analogous to a buffer. This buffer (empty glass) can store 33 cl. of liquid (data). Now, imagine that I wish to transfer a pint, which is about 47 cl., of beer from my full pint glass into the empty 33 cl. glass. As I begin to fill the glass (buffer) with beer (data), everything is fine until the end when beer begins to spill over the glass and onto the table. This is an example of an overflow. Clearly, such an overflow is bad for beer and unfortunately, even worse if such vulnerabilities exist in computer programs.

When the data exceeds the size of the buffer, the extra data can overflow into adjacent memory locations, corrupting valid data and possibly changing the execution path and instructions. The ability to exploit a buffer overflow allows one to possibly inject arbitrary code into the execution path. This arbitrary code could allow remote, system-level access, giving unauthorized access to not only 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; second generation overflows involve heaps, function pointers, and off-by-one exploits; and finally, third generation overflows involve format string attacks and vulnerabilities in heap structure management.

For simplicity, the following explanations will assume an Intel CPU architecture, but the concepts can be applied to other processor designs.

**First generation**

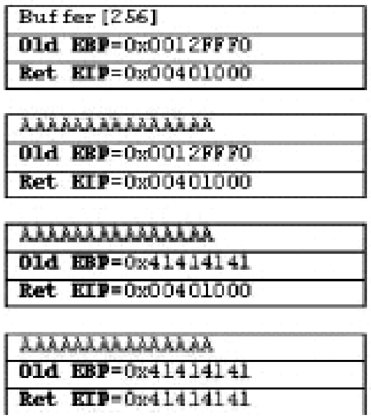
First generation buffer overflows involve overflowing a buffer that is located on the stack.

**Overflowing a Stack Buffer**

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).

```
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
}
```

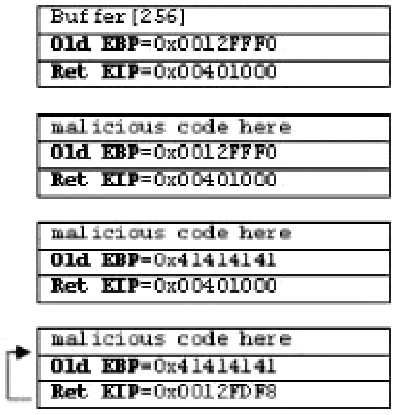
The diagram below illustrates how the EIP (where to execute next) is modified due to the program overflowing the small 256 byte buffer. When data exceeds the size of the buffer, the data overwrites adjacent areas of data stored on the stack including critical values such as the instruction pointer (EIP),which define the execution path. By overwriting the return EIP, an attacker can change what the program should execute next.



1. A function is using a buffer 256 bytes long. The program attempts to fill the buffer with 512As.
2. After 256 As the buffer is full and remaining as will begin too overflow into adjacent memory.
3. The remaning as being to overwrite the od EBP.
4. And also overwrite the return BP

**Exploiting a stack buffer**

Overflow Instead of filling the buffer full of As, a classic exploit will fill the buffer with its own malicious code. Also, instead of overwriting the



1. A function is using a buffer 256 bytes long. The program beings to fill the buffer with attacker code.
2. After 256 bytes, the buffer is full any remaining bytes will being to overflow into adjacent memory.
3. First EPB to overwritten
4. And teh EIP is overwritten with the address pointing back to the malicious code. Now, the program will being to execute the maliclous code.

return EIP (where the program will execute next) with random bytes, the exploit will overwrite EIP with the address to the buffer, which is now filled with malicious code.

This causes the execution path to change and causes the program to execute injected malicious code. While the above example demonstrates a classic stack-based (first generation) buffer overflow, there are variations. The exploit utilized by CodeRed was a first generation buffer overflow that is more complex and is described below.

**Causes of stack-based overflow vulnerabilities**

Stack-based buffer overflows are caused by programs that do not verify the length of data being copied into a buffer. This is often caused by using common functions that do not limit the amount of data that is copied from one location to another.

**Current Security**

Many of these blended threats are effective today because most current security products can not prevent the threats. Furthermore, only some products can detect the threats once they have arrived on the system, or simply alert an attack has taken place. For example, traditional antivirus products do not scan Windows memory. This means blended threats such as CodeRed, which reside solely in memory might remain undetected. In addition, most intrusion detection products merely alert rather than block when a signature matches. Thus, while an administrator may receive an alert, the worm will have already infected the machine. Some modern intrusion detection systems employ gated technology, which means threats are actually blocked when detected. One of our own solutions included the use of a memory scanner to detect the worm in memory. The program also determined the need for the patch. Unfortunately, CodeRed's code can appear on the heap of IIS processes that are not vulnerable to the attack. Therefore a scanning solution had to prove the active existence of the worm on the machine

## REFERENCES

1. [spaff88] Eugene H. Spafford, The Internet Worm Program: An Analysis.
2. [Gordon88] Sarah Gordon, "The Worm Has Turned," Virus Bulletin, August, 1998.
3. [McCorkendale-Szor] "Code REd Buffer Overflow," Virus Bulletin, September 2001.
4. http://www.cve.mitre.org/, (Common Vulnerabilities and Exposures).
5. http://www.cert.org/.
6. [Szor99] "Memory Scanning Under Windows NT," Int. Virus Bull. Conf., 1999, pp.325_346, also see
7. http://securityresponse.symantec.com/avcenter/reference/memory. scanning. winnt.pdf.
8. [Szor2000] "Bolzano Bugs NT," Virus Bulletin, September 1999.
9. [Litchfield] Windows 2000 Format String Vulnerabilities, May 8, 2002.
10. [Howard-LeBlanc] Writing Secure Code, Chapter 12.
11. [Personal Communication] Bruce McCorkendale, Frederic Perriot.
12. [One] Smashing The Stack For Fun And Profit, Phrack 49, Vol. 7, Issue #49, File 14.
13. http://www.peterszor.com/badtrans.pdf "Bad Transfer," Virus Bulletin, February 2002.
14. http://msdn.microsoft.com/workshop/components/activex/safety.asp.
15. http://msdn.microsoft.com/library/en-us/script56/html/letcreatetypelib.asp.