White Paper April 2005

McAfee System Protection Solutions

Buffer Overflow

Exploits: The Why and How

www.mcafee.com

Table of Contents

Buffer overflow exploits are the tool of choice of today's attacker. These exploits have the most power, are the easiest to use, and are all too common. Buffer overflows constitute the largest single threat to enterprises today.

- Buffer overflow exploits are very common. There are hundreds of known unchecked buffers that can be overflowed by hackers with more being discovered all the time. Over 60% of CERT advisories deal with buffer overflow exploits.
- Buffer overflow exploits are easy to use. Nearly anyone (12 year olds and script kiddies included) can download buffer overflow attack code and follow a simple "recipe" to execute it. No advanced technical knowledge is necessary to run pre-written buffer overflow exploit code.
- Buffer overflow exploits are very powerful. In many cases, the malicious code that executes as a result of a buffer overflow will run with administrator-level privileges, and therefore can do anything it wants to the server.

Unchecked buffers

Unchecked buffers enable buffer overflows. Unchecked buffers occur when the programmers who write an application neglect to check data size. In most commonly-used programming languages, buffers to store data are allocated with a fixed size. Programmers allocate enough buffer space to store "normal" data. Let's take an example:

Assume for a moment that you are a programmer in charge of writing a login routine for an application. Your routine must accept a user name and password from the user and determine whether or not to allow the user access. One of the first things you must determine is how much buffer space to allocate for the user name and password buffers. Given that the maximum length of any user name for this system is 32 characters, you decide to allocate 100 characters of buffer memory, which is more than enough to hold the longest username.

You may erroneously think that since the 100 character buffer is much larger than the 32 characters that a user would normally submit, this routine is not vulnerable to buffer overflow exploits. However, because you neglected to check the length of the data submitted before copying it into the buffer, this routine can be exploited. An attacker only needs to send more than 100 characters to this routine, and the buffer will overflow.

Exploiting the overflow

In the simplest case, an overflowed buffer will cause the application that owns that buffer to crash, which results in denial-of-service. However, in most cases, attackers can do much more. By cleverly constructing the data submitted, attackers can cause their arbitrary attack code to be executed. This attack code can do nearly anything the attacker desires, from reformatting the hard drive, to stealing data, installing backdoor programs, etc.

Computer programs are organized into subroutines. The program's main routine calls each subroutine, which performs its particular function(s) and then returns control to the main routine.

Each subroutine, in turn, has to save various pieces of information in order to perform its work. Subroutines use an area of memory called the stack for storing this information. One of these pieces of information is the memory address to which the subroutine should return control when it is finished with its work.

Subroutines also store temporary data (buffers) on the stack. Each time a subroutine is run, the required memory is allocated on the stack in a unit called a stack frame. This stack frame includes space for any buffers the subroutine requires, as well as the calling routine's return address. When the subroutine completes its work, it returns control to the calling routine by jumping to the address stored in the stack frame, and the stack frame is deleted.

All of this machinery works perfectly until a stack buffer overflows. When a user sends 1000 characters to a 100 character stack buffer, the extra 900 characters overwrite adjacent memory in the stack frame, overwriting other buffers and the stack frame's return address.

Now, when the subroutine attempts to return control to the main program, it jumps to the address that is stored in the return address portion of the stack frame. Unfortunately, this address has been overwritten by the overflowed buffer and the address is corrupted. When the program tries to jump to a nonexistent address, the program crashes.

Now, for the clever part

If the attacker sends 1000 characters that are carefully chosen, he or she can control the return address. Rather than jumping to a non-existent address, the attacker can instruct the program to jump to the address of malicious exploit code. There are two tasks that must be performed to accomplish

this:

1. Loading the malicious code
2. Executing the malicious code

Overflowing a stack buffer achieves both of these tasks. The attacker sends a very long string of input data to the program. The input data includes the malicious code as well as the address of that code. When this input data overflows the stack, the malicious code is loaded into stack memory, and the subroutine's return address is overwritten to point to that malicious code. When the subroutine terminates, the program jumps to the malicious code, and that code is executed. The results of executing the attacker's malicious code could be catastrophic. If the malicious code reformats the system's hard drive, crucial data may be lost, and significant time will be wasted rebuilding the system. Just as easily, the malicious code could attack other machines, install backdoors, steal passwords, or any number of other possibilities. The malicious code runs under the process context of the application it is attacking. Thus, if the application has root- or administrator-level privileges, the code will run as root and be able to execute any command. If the application is not running as root, the attacker can still use a buffer overflow exploit to load a privilege escalation exploit which would then give the desired root privileges. Thus, buffer overflow exploits are very useful to attackers.

Other types of buffer overflow exploits

The stack-based overflow technique discussed here is the most common type of buffer overflow exploit. However, other techniques are gaining popularity. Heap-based overflows act similarly to stack-based overflows, but overflow buffers on the heap. Return into- libc exploits use buffer overflows on the stack or heap to cause execution of code in the system's own libc library. These newer techniques provide new avenues of buffer overflow attack that are more and more difficult to detect and prevent.

Preventing buffer overflow exploits

Buffer overflow exploits can be prevented. If programmers were perfect, there would be no unchecked buffers, and consequently, no buffer overflow exploits. However, programmers are not perfect, and unchecked buffers continue to abound. When unchecked buffers are found, vendors often release patches that correct the problem. Unfortunately, keeping patches up to date on a large numbers of systems is difficult and many system administrators fall behind in patch deployment.

Another way to prevent buffer overflow exploits is to use McAfee Entercept. Along with protecting the operating system, applications, and data, Entercept contains patented technology that blocks the execution of code from overflowed buffers. By doing this, Entercept prevents systems from being compromised by buffer overflow exploits. This protection provides a layer of security that largely eliminates the most significant threat to servers: buffer overflow exploits. System administrators should still patch their systems, but using Entercept buys the administrator crucial time to test and deploy patches while Entercept protects them from buffer overflow threats. Entercept prevents all three major types of buffer overflow exploits: stack-based, heap-based, and return-into-libc. It is the only security product on the market that can block all of these exploit types.

Summary

Buffer overflow exploits are here to stay. They are pervasive, powerful, and easy to use. They are the tool of choice to today's attacker, and must be prevented. Keeping systems up-to-date with the most current security patches and using McAfee Entercept will protect servers against these powerful threats.

McAfee, Inc. 3965 Freedom Circle, Santa Clara, CA 95054