

## Understanding Technical Vulnerabilities: Buffer Overflow Attacks

By Michael Legary, CISSP

Deep down in the source code of every computer program lay countless exposures waiting to be found by unscrupulous individuals. By exploiting tiny programmer oversights in program code, attackers can bypass access controls, audit logs, and a myriad of other security measures protecting corporate systems in a matter of minutes.

The most common code-based attack is the buffer overflow. Buffer overflow attacks are application- and system-specific programs, called exploits, designed to target poorly utilized procedure calls in software program code. Buffer overflow attacks are written for specific operating systems or applications and typically require custom changes to the exploit code, depending on the hardware platform being targeted. In most cases, buffer overflow attacks allow complete uncontrolled access to systems and the precious confidential business information they contain.

The goal of a buffer overflow attack is to overwrite sections of memory with specific commands that the attacker wants executed. These instructions typically include functions that allow them to create denial-of-service conditions or gain remote access to high-level user and system accounts. This is possible because the typical software program has a large number of data inputs and function calls, each of which is potentially vulnerable to a buffer overflow exploit. Even though there has been an enhanced awareness of buffer overflow attacks through out the software development industry and the risks buffer overflows present to organizational security, no one solution completely protects systems from exploitation.



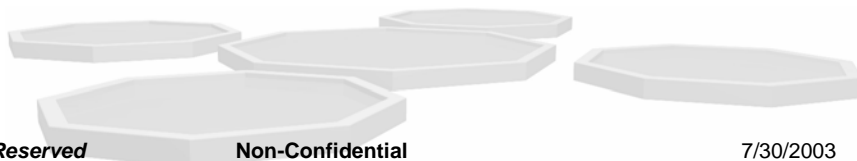
Michael Legary CISSP, GCIH, CISM  
Operations Director  
Securis

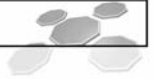
Michael is Operations Director of Securis Inc, providing managed security, information system security consulting and investigation services to a range of North American clients. He specializes in incident handling, intrusion detection, security assessments, and Investigations. Michael is membership director for the Information Protection Association of Manitoba (IPAM) and is a prominent member of the information security community in Canada.

### UNIDENTIFIED ORGANIZATIONAL RISK

Organizations unwittingly expose themselves to an unnecessary degree of risk when they fail to review software for buffer overflow issues. Both vendor and internally developed software must be reviewed for proper implementation of required safeguards and protection against known vulnerabilities before integration into the IT environment to effectively mitigate risk.

One of the largest risks is integrating vendor software into the information environment before reviewing the development standards used and the integrated application security controls. In reality, the problem goes beyond vendor-developed software; custom-developed software is just as likely to be a major point of exposure for an organization. These exposures tend to exist because many organizations lack secure coding procedures. Furthermore, the focus of internal software development is often based on highly constrained, cost-conscious business solution development rather than on particularly "secure" solution development. We must remain cognizant of these software development issues, be aware of technical code vulnerabilities and remain continually vigilant to ensure systems are audited for the existence of insecure code.





## HOW DO BUFFER OVERFLOW ATTACKS WORK?

A buffer overflow attack takes advantage of poorly implemented program functions that move data to and from system memory addresses. The most common type of buffer overflow attack exploits insufficient boundary checking around data input or manipulation within a program. Without proper boundary checking within a program, attackers can create strings that are longer than the intended input length and redirect the program's response so it returns to a specific memory address where the attacker's own program instructions have been placed.

A buffer overflow technique that has become increasingly popular with attackers is based on a vulnerability relating to sending malformed URL requests to a web server. When the web server receives the URL request, it loads the URL into a memory buffer. A memory buffer is an allocated space in memory reserved for use by a specific program variable. The vulnerability is exploited when the program does not check to ensure that the size of the URL can fit within the buffer allocated for the request. As a direct result buffers are overflowed corrupting the execution flow of the web server program.

Think of a memory buffer as being similar to water in a bucket. A bucket can hold water, but if you pour too much water in, it will spill out of the bucket and onto the floor. Much like a bucket of water, a buffer in memory is designed to hold only a certain amount of information. If you try to place more characters in the buffer than what was originally intended, the extra information will spill over into other buffers in memory. This process of overflowing memory buffers has been called "smashing the stack."

## AN EXAMPLE SCENARIO

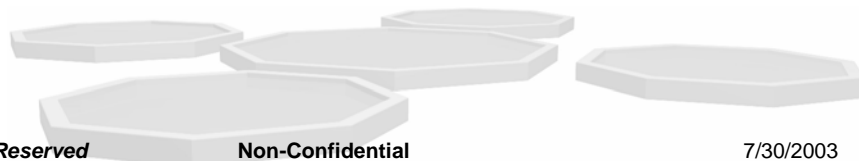
How can we identify what types of program code are susceptible to buffer overflows? What does a buffer overflow look like?

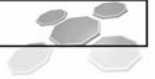
The sample "C" language program named Example.c in **Figure 1** illustrates how a buffer overflow works. We will walk through the lines of interest, highlighted in red, in the following paragraphs.

### *Example.c*

```
#include <stdio.h>  
int main(int argc, char **argv) {  
    char small [4] = "SSS";  
    char big [8] = "BBBBBBB";  
  
    strcpy (big, "XXXXXXXXXX");  
    printf ("%s\n", small);  
return 0; }
```

Figure 1





```
char small [4] = "SSS";  
char big [8] = "BBBBBBB";
```

**Figure 2**

The first point of interest occurs when the two lines of code in **Figure 2** are executed and the program completes the following steps:

1. Creates a buffer called "small" that can hold up to four characters.
2. Places three letter "S's" into the "small" buffer.
3. Creates a buffer called "big" that can hold eight characters.
4. Places seven letter "B's" into the "big" buffer.

When the program has completed running these two lines of code, a snapshot of the system memory, shown in **Figure 3**, will show us the contents of the "small" and "big" buffers, with their end place in memory denoted by a null character (/0).

\0	S	S	S
\0	B	B	B
B	B	B	B

**Initial Stack**

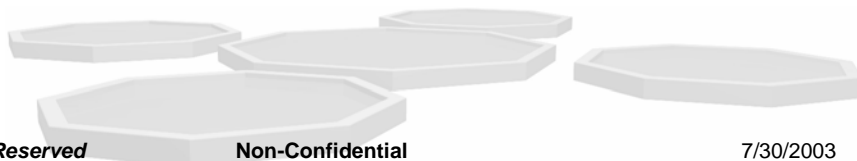
**Figure 3**

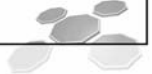
After completing the first two lines of code the program executes the instructions illustrated in **Figure 4** which includes the following steps:

```
strcpy (big, "XXXXXXXXXX");  
printf ("%s\n", small);
```

**Figure 4**

5. The program places 10 letter "X's" into the "big" buffer.
6. Prints the contents of the "small" buffer.





\0	\0	X	X
X	X	X	X
X	X	X	X

## After the overflow

Figure 5

Once the "strcpy" function has been executed, the contents of the buffer's memory will be greatly changed. **Figure 5** shows that 10 letter "X"s have been placed in memory, but by doing so; the contents of the "small" buffer have been completely destroyed. This happened because the "strcpy" function did not check to ensure the data would fit into the buffer before completing the function call. This is the essence of a buffer overflow attack.

## EXPLOITING A BUFFER OVERFLOW

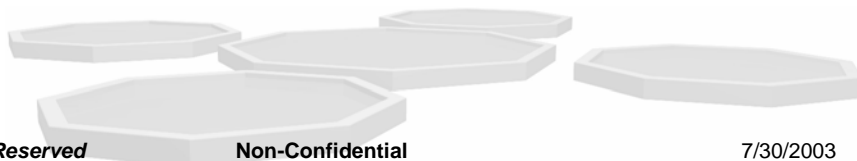
Attackers find buffer overflows in programs by utilizing several methods. By running an identified program through a code inspection tool, such as a debugger, an attacker can locate possible system calls and functions that do not restrict the length and type of input allowed into the program. System calls can also be identified by reviewing the program source code by hand, looking for potentially vulnerable calls and functions. Finally, attackers can simply use brute force to attempt to find weaknesses in the program code. This is accomplished by trying to force large amounts of text into program input fields, causing the program to crash. By reviewing the memory dump from the program crash, they can identify possible vulnerabilities.

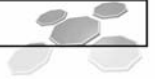
Certain programming functions, if not handled correctly, can cause buffer overflow vulnerabilities. **Figure 6** lists C language functions that are vulnerable to buffer overflows:

strcpy	fgets
strncpy	gets
strcat	getws
sprintf	memcpy
scanf	memmove

Figure 6

Once attackers have identified a buffer overflow vulnerability, they can create custom input strings that are longer than the known input length of the buffer. These custom input strings contain machine code the attackers want to execute on the system. The machine code contains system commands that, once sent, ensure that the program will not crash; instead, the code will direct the program to complete additional instructions contained within the same string. These additional instructions contain the code that gives the attacker remote access and a backdoor into the system.





## HOW TO PROTECT AGAINST BUFFER OVERFLOWS

Organizations can best protect themselves against buffer overflow attacks during the design phase of the systems development life cycle (SDLC). During the design phase, developers can define specifications regarding acceptable implementation of procedure calls for data input and manipulation, as well as add secure error-exception handling requirements into the program design. Additionally, audits completed at regular intervals throughout the SDLC can ensure that only defined and accepted procedures and controls are implemented during development.

Recently the .NET framework - a set of development tools for building ASP Web applications, XML Web services, desktop applications, and mobile applications - has implemented the ability to catch runtime errors and exceptions within certain programming languages, such as C# and C++. This functionality will allow for the detection of buffer overflow vulnerabilities within a program during the development phase of the SDLC. It is important to note that in order to catch these exceptions the programmers must explicitly add these protective measures into the code to detect possible buffer overflow attacks.

If buffer overflow vulnerabilities exist after code has already been compiled and released, the best way to protect against exploit is to identify the vulnerable sections of code, make the required changes to prevent overflows from occurring, and recompile the code. If recompiling is not an option, or if the program contains a substantial number of critical vulnerabilities that require immediate remediation, security products called "stack guards" can be used to protect against buffer overflow exploits. Although an operating system stack guard or "non-exec patch" does not offer 100 percent protection, they can add an additional layer of protection to the application security model.

## DUE CARE REQUIRED

Buffer overflow vulnerabilities exist in a significant portion of software used within organizations. With minimal effort, attackers can exploit these numerous vulnerabilities by creating targeted attacks against these organizations, allowing those attackers to leak sensitive information or prevent legitimate use of business-critical programs. Organizations must take due care to ensure proper awareness exists within information systems management, technical and audit functions regarding code based vulnerabilities. We should ensure comprehensive secure code development standards exist, secure controls are defined and audit procedures are in place to protect against and detect buffer overflow vulnerabilities.

