WHAT IS A BUFFER OVERFLOW? LEARN ABOUT BUFFER OVERRUN VULNERABILITIES, EXPLOITS & ATTACKS

A buffer overflow, or buffer overrun, is a common software coding mistake that an attacker could exploit to gain access to your system. To effectively mitigate buffer overflow vulnerabilities, it is important to understand what buffer overflows are, what dangers they pose to your applications, and what techniques attackers use to successfully exploit these vulnerabilities.

- This error occurs when there is more data in a buffer than it can handle, causing data to overflow into adjacent storage.
- This vulnerability can cause a system crash or, worse, create an entry point for a cyberattack.
- C and C++ are more susceptible to buffer overflow.
- Secure development practices should include regular testing to detect and fix buffer overflows. These practices include automatic protection at the language level and bounds-checking at run-time.
- Veracode's binary SAST technology identifies code vulnerabilities, such as buffer overflow, in all code — including open source and third-party components —so that developers can quickly address them before they are exploited.

Definition of a Buffer Overflow

A buffer is a sequential section of memory allocated to contain anything from a character string to an array of integers. A buffer overflow, or buffer overrun, occurs when more data is put into a fixed-length buffer than the buffer can handle. The extra information, which has to go somewhere, can overflow into adjacent memory space, corrupting or overwriting the data held in that space. This overflow usually results in a system crash, but it also creates the opportunity for an attacker to run arbitrary code or manipulate the coding errors to prompt malicious actions.

Many programming languages are prone to buffer overflow attacks. However, the extent of such attacks varies depending on the language used to write the vulnerable program. For instance, code written in Perl and JavaScript is generally not susceptible to buffer overflows. However, a buffer overflow in a program written in C, C++, Fortran or Assembly could allow the attacker to fully compromise the targeted system.

Cybercriminals exploit buffer overflow problems to alter the execution path of the application by overwriting parts of its memory. The malicious extra data may contain code designed to trigger specific actions — in effect sending new instructions to the attacked application that could result in unauthorized access to the system. Hacker techniques that exploit a buffer overflow vulnerability vary per architecture and operating system.

Coding errors are typically the cause of buffer overflow. Common application development mistakes that can lead to buffer overflow include failing to allocate large enough buffers and neglecting to check for overflow problems. These mistakes are especially problematic with C/C++, which does not have built-in protection against buffer overflows. Consequently, C/C++ applications are often targets of buffer overflow attacks.

Buffer Overflow Attack Example

[Adapted from “Buffer Overflow Attack Explained with a C Program Example,” Himanshu Arora, June 4, 2013, *The Geek Stuff*]

In some cases, an attacker injects malicious code into the memory that has been corrupted by the overflow. In other cases, the attacker simply takes advantage of the overflow and its corruption of the adjacent memory. For example, consider a program that requests a user password in order to grant the user access to the system. In the code below, the correct password grants the user root privileges. If the password is incorrect, the program will not grant the user privileges.

```
printf ("\n Correct Password \n");
pass = 1;
}
if(pass)
{
/* Now Give root or admin rights to user*/
printf ("\n Root privileges given to the user \n");
}
return 0;
```

However, there is a possibility of buffer overflow in this program because the gets() function does not check the array bounds.

Here is an example of what an attacker could do with this coding error:

```
$ ./bfrovrflw
Enter the password :
hhhhhhhhhhhhhhhhhhhh
Wrong Password
Root privileges given to the user
```

In the above example, the program gives the user root privileges, even though the user entered an incorrect password. In this case, the attacker supplied an input with a length greater than the buffer can hold, creating buffer overflow, which overwrote the memory of integer “pass.” Therefore, despite the incorrect password, the value of “pass” became non zero, and the attacker receives root privileges.

To prevent buffer overflow, developers of C/C++ applications should avoid standard library functions that are not bounds-checked, such as gets, scanf and strcpy.

In addition, secure development practices should include regular testing to detect and fix buffer overflows. The most reliable way to avoid or prevent buffer overflows is to use automatic protection at the language level. Another fix is bounds-checking enforced at run-time, which prevents buffer overrun by automatically checking that data written to a buffer is within acceptable boundaries.

Veracode Helps Identify Buffer Overflows

Veracode’s cloud-based service identifies code vulnerabilities, such as buffer overflow, so that developers can address them before they are exploited.

Unique in the industry, Veracode’s patented binary static application security testing (SAST) technology analyzes all code — including open source and third-party components — without requiring access to source code.

SAST supplements threat modeling and code reviews performed by developers, finding coding errors and omissions more quickly and at lower cost via automation. It’s typically run in the early phases of the software development lifecycle because it’s easier and less expensive to fix problems before going into production deployment.

SAST identifies critical vulnerabilities such as SQL injection, cross-site scripting (XSS), buffer overflows, unhandled error conditions and potential back-doors. In addition, our binary SAST technology delivers actionable information that prioritizes flaws according to severity and provides detailed remediation information to help developers address them quickly.

https://www.veracode.com/security/buffer-overflow