Buffer Overflow

A buffer is a temporary area for data storage. When more data (than was originally allocated to be stored) gets placed by a program or system process, the extra data overflows. It causes some of that data to leak out into other buffers, which can corrupt or overwrite whatever data they were holding.

In a buffer-overflow attack, the extra data sometimes holds specific instructions for actions intended by a hacker or malicious user; for example, the data could trigger a response that damages files, changes data or unveils private information.

Attacker would use a buffer-overflow exploit to take advantage of a program that is waiting on a user's input. There are two types of buffer overflows: stack-based and heap-based. Heap-based, which are difficult to execute and the least common of the two, attack an application by flooding the memory space reserved for a program. Stack-based buffer overflows, which are more common among attackers, exploit applications and programs by using what is known as a stack: memory space used to store user input.

Let us study some real program examples that show the danger of such situations based on the C.

In the examples, we do not implement any malicious code injection but just to show that the buffer can overflow. Modern compilers normally provide overflow checking option during the compile/link time but during the run time it is quite difficult to check this problem without any extra protection mechanism such as using exception handling.

```c
// A C program to demonstrate buffer overflow
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
	// Reserve 5 byte of buffer plus the terminating NULL.
	// should allocate 8 bytes = 2 double words,
	// To overflow, need more than 8 bytes...
	char buffer[5]; // If more than 8 characters input
	// by user, there will be access
	// violation, segmentation fault
	// a prompt how to execute the program...
	if (argc < 2)
	{
			printf("strcpy() NOT executed....\n");
			printf("Syntax: %s <characters>\n", argv[0]);
			exit(0);
	}
	// copy the user input to mybuffer, without any
	// bound checking a secure version is srtcpy_s()
	strcpy(buffer, argv[1]);
	printf("buffer content= %s\n", buffer);

	// you may want to try strcpy_s()
	printf("strcpy() executed...\n");

	return 0;
}
```

```
Input  : 12345678 (8 bytes), the program run smoothly.
Input : 123456789 (9 bytes)
"Segmentation fault" message will be displayed and the program terminates.
```

The vulnerability exists because the buffer could be overflowed if the user input (argv[1]) bigger than 8 bytes. Why 8 bytes? For 32 bit (4 bytes) system, we must fill up a double word (32 bits) memory. Character (char) size is 1 byte, so if we request buffer with 5 bytes, the system will allocate 2 double words (8 bytes). That is why when you input more than 8 bytes; the mybuffer will be over flowed

Similar standard functions that are technically less vulnerable, such as strncpy(), strncat(), and memcpy(), do exist. But the problem with these functions is that it is the programmer responsibility to assert the size of the buffer, not the compiler.

Every C/C++ coder or programmer must know the buffer overflow problem before they do the coding. A lot of bugs generated, in most cases can be exploited as a result of buffer overflow.

Buffer Overflow and Web Applications

Attackers use buffer overflows to corrupt the execution stack of a web application. By sending carefully crafted input to a web application, an attacker can cause the web application to execute arbitrary code – effectively taking over the machine.

Buffer overflow flaws can be present in both the web server or application server products that serve the static and dynamic aspects of the site, or the web application itself. Buffer overflows found in widely used server products are likely to become widely known and can pose a significant risk to users of these products. When web applications use libraries, such as a graphics library to generate images, they open themselves to potential buffer overflow attacks.

Buffer overflows can also be found in custom web application code, and may even be more likely given the lack of scrutiny that web applications typically go through. Buffer overflow flaws in custom web applications are less likely to be detected because there will normally be far fewer hackers trying to find and exploit such flaws in a specific application. If discovered in a custom application, the ability to exploit the flaw (other than to crash the application) is significantly reduced by the fact that the source code and detailed error messages for the application are normally not available to the hacker.

Consequences

Availability: Buffer overflows generally lead to crashes. Other attacks leading to lack of availability are possible, including putting the program into an infinite loop.

Access control (instruction processing): Buffer overflows often can be used to execute arbitrary code, which is usually outside the scope of a program's implicit security policy.

Other: When the consequence is arbitrary code execution, this can often be used to subvert any other security service.

Exposure period

Requirements specification: The choice could be made to use a language that is not susceptible to these issues.

Design: Mitigating technologies such as safe-string libraries and container abstractions could be introduced.

Implementation: Many logic errors can lead to this condition. It can be exacerbated by lack of or mis¬use of mitigating technologies.

Environments Affected

Almost all known web servers, application servers, and web application environments are susceptible to buffer overflows, the notable exception being environments written in interpreted languages like Java or Python, which are immune to these attacks (except for overflows in the Interpreter itself).

Platform

Languages: C, C++, Fortran, Assembly

Operating platforms: All, although partial preventative measures may be deployed, depending on environment.

How to Determine If You Are Vulnerable

For server products and libraries, keep up with the latest bug reports for the products you are using. For custom application software, all code that accepts input from users via the HTTP request must be reviewed to ensure that it can properly handle arbitrarily large input.

How to Protect Yourself

Keep up with the latest bug reports for your web and application server products and other products in your Internet infrastructure. Apply the latest patches to these products. Periodically scan your web site with one or more of the commonly available scanners that look for buffer overflow flaws in your server products and your custom web applications. For your custom application code, you need to review all code that accepts input from users via the HTTP request and ensure that it provides appropriate size checking on all such inputs. This should be done even for environments that are not susceptible to such attacks as overly large inputs that are uncaught may still cause denial of service or other operational problems.