Programmer's Points to Remember:

- ✓ Always do bounds checking on arrays.
- ✓ Always do bounds checking on pointer arithmetic.
- ✓ Before you copy to, format, or send input to a buffer make sure it is big enough to hold whatever might be thrown at it.
- ✓ Remember that the input itself might cause a buffer overflow and not the size of the input. One integer can create a buffer overflow.
- ✓ Be aware of unsafe library functions. Know how to use them correctly.
- ✓ Be careful of off-by-one errors. A buffer of 512 bytes can hold a string of 511 characters and one NULL character. Library functions like **fgets()** and **scanf()** expect the number of characters in the string, not the number of characters in the buffer, so you need to put 511 for a 512-byte buffer.
- ✓ Remember that an array declared in C as **int A[100]** can only be accessed with indices **A[0]** through **A[99]**.
- ✓ Be paranoid about old code. Be careful using international character sets with old code.
- ✓ Challenge all of your assumptions like an attacker would.
- ✓ Assume that ALL buffer overflows are a security risk.

Code Inspectors' Points to Remember:

- ✓ Look for bounds checking on arrays.
- ✓ Look for bounds checking on pointer arithmetic.
- ✓ Make sure a buffer is big enough to hold whatever might be thrown at it before it is copied to, formatted, or sent input of any kind.
- ✓ Remember that the input itself might cause a buffer overflow and not the size of the input. Code inspection is more likely to catch these kinds of errors than testing.
- ✓ Identify all library function calls and make sure they are safe. Static analyzers are good tools for code inspection of unsafe library function calls.
- ✓ Watch for off-by-one errors.
- ✓ Remember that an array declared in C as **int A[100]** can only be accessed with indices **A[0]** through **A[99]**.
- ✓ Include old code in code inspection, even if you inspected it before.
- ✓ Challenge all assumptions like an attacker would.
- ✓ Assume that ALL buffer overflows are a security risk.

Tester's Points to Remember:

- ✓ Check all string and buffer inputs by entering a very long string or too much data. If other data is corrupted or the program crashes you probably have a buffer overflow.
- ✓ Remember that the input itself might cause a buffer overflow and not the size of the input. Try the maximum and minimum values for numeric inputs and divide them up into partitions where if you test one number in the partition the software should work correctly for all numbers in that partition.
- ✓ For each partition of value or size on any input, try the numbers on the ends of the partition, and also these numbers plus/minus one.
- ✓ Test old code when you're using it for new things, even if you tested it before. If your software allows the user to use UNICODE then do all of the testing you did for ASCII with UNICODE as well.
- ✓ Test code on all platforms it was meant to run on.
- ✓ Challenge all assumptions like an attacker would.
- ✓ Assume that ALL buffer overflows are a security risk.

Unsafe Library Function Calls Points to Remember:

- ✓ Never, ever, ever use **gets()**. Use **fgets()** with **stdin** as the file instead. **fgets()** will let you specify a string size, but put the size of the string not the size of the buffer or you will get an off-by-one error.
- ✓ Avoid **strcpy()** and **strcat()**. Use **strncpy()** and **strncat()** instead. **strcpy()** and **strcat()** are okay with constant strings for the source, but it's probably not safe to assume that a string will always be constant.
- ✓ Use precision specifiers with the **scanf()** family of functions (**scanf()**, **fscanf()**, **sscanf()**, etc.). Otherwise they will not do any bounds checking for you. Watch for off-by-one errors.
- ✓ Never use variable format strings with the **printf()** family of functions.
- ✓ Every file or path handling library function has its own quirks, so be careful no matter what operating system you're programming in.
- ✓ Watch for off-by-one errors with otherwise safe functions such as **memcpy()**.
- ✓ When using **streadd()** or **strecpy()**, make sure the destination buffer is four times the size of the source buffer.
- ✓ Be careful of EVERY library function that takes a pointer (especially string pointers) as input for a place to store its output. If there is not another parameter for the buffer size or string size then that function does <u>not</u> do bounds checking.
- ✓ You should also be careful and do bounds checking when using functions like **getc()** and **read()** in a loop.
- ✓ Functions like **syslog()** and **getopt()** take strings as input but, depending on the implementation, might not check the size of the string before using it. You should truncate strings to a reasonable length before passing pointers to them as input to any library function.
- ✓ When using library functions that take buffer size as a parameter, make sure your buffer is as big as you say it is.