String Format Attack

Chetan Mistry

October 31, 2019

Contents

1 Abstract

A string format attack works by exploiting vulnerabilities in source code to examine and/or modify the values stored in variables.

2 String Formatting in C

Some String manipulation functions in C have vulnerabilities which can be exploited by an attacker. An example of this is the function "scanf" which allows the user to provide an arbitrary input.

2.1 scanf()

"scanf()" is a function which reads from stdin and stores it into a variable. This input can be anything (strings, numbers, etc), and can have any length.

2.2 printf()

"printf()" is a function which takes in a string and any parameters to that string (e.g variables) and will output to stdout. The variables can be printed by using special tokens in the string which will map to specific output formats, for example "%d" outputs a number in denary format, whereas "%x" outputs in hex. The way this works is that the variables to be printed are placed into the function stack, and then whenever a token is encountered, it simply replaces it with whatever is at the current stack pointer, and then moving the stack pointer so that it is now at the next variable.

2.3 "%n"

"%n" is a special formatting character which, instead reading the value stored at the stack pointer, will instead overwrite it with the number of bytes read in so far.

3 Attacks

3.1 Denial of Service (DoS)

"%s" is interpreted by printf() as a command which will use the current value in function stack as a pointer to a null-terminated string. This means that the program will unconditionally dereference a value, this leads to 2 different situations:

1. The address is valid and accessible to the program, in which

case it will continually print the characters found at that address until the null-character (\0) is met.

1. The address is invalid/out-of-bounds to the current program,

resulting in a segmentation-fault (seg-fault), which causes the OS to terminate execution.

Denial of Service works by entering enough "%s" characters until the value in the stack is interpreted as an invalid pointer.

An example input to cause the second situation is "%s%s%s%s%s%s%s%s%s" This continually moves the stack pointer, dereferencing the values and printing what is stored in those addresses.

Another DoS attack can be done by what is known as "Stack Smashing" which is when a large number of formatting characters are input (eg. %d, %x, %c, etc.), if enough of these formatting characters are entered, the stack pointer will move out of the function call stack.

3.2 Reading Values on the Stack

When printf() is called, it loads the values to be printed into its call stack. These values are then read off every time the string formatting characters are met. It is possible to read values which aren't in the printf() call stack by moving the stack pointer outside of the scope of the function by continually entering the format characters. This then allows access to variables stored by the program.

3.2.1 Variables vs Pointers

Some variables are statically allocated in the program stack at compile time (eg char[6]), these values can be directly output by using specific formatting characters (%x).

Other variables are dynamically allocated (char*) in the program heap, and can only be accessed by dereferencing a pointer to them. To print them, specific formatting characters (%s) must be used as they interpret the value on the stack as an address rather than a value.

3.2.2 Data Structures

Data Structures are typically dynamically allocated, as a result the pointer to them will only point to the first value stored. In order to print off the whole structure, we can utilise the fact that data is stored in contiguous blocks of memory. This means that if an attacker knows the address of the first element, it is possible to calculate the addresses of further variables by noting the type of the data structure (eg int, char, bool, etc) and using the size of the type as the size to step through in memory.

3.3 Writing Values to the Stack

This attack utilises the "%n" operator described earlier. If an attacker wants to change the value stored in a particular location in memory, then the

address must first be found in the program stack, then "%n" can be used to change the value.