**Sploit 1 description**

The memory for variables unsigned I, len and buffer are allocated on the stack after the processor “prolog” with the following assembly code:

sub $0xbd8,%esp

Just beside this memory lies the Saved Frame Pointer (SFP) and the return address RET.

We find the address of the buffer in gdb by entering the following commands:

print $ebp - 0xbd8

result: (void \*) 0xffbfd060

We can hence overwrite the RET by overflowing the buffer as no checks are in place. However, the variable “i” is used as a counter in the copyFile function. Suppose we were to fill our buffer with the value “A” 3016 times.

i **=** 0**;**

c **=** fgetc**(**source**);**

**while** **(**c **!=** EOF**)** **{**

buffer**[**i**]** **=** **(**unsigned char**)** c**;**

c **=** fgetc**(**source**);**

i**++;**

**}**

On paper we would be overwriting the RET address. However, the variable “i” is stored above buffer. By overflowing the buffer, we would be inadvertently overwriting the LSB of “i” with the value A which would change the counter preventing the buffer from being overwritten.

In order to get past the variable “i”, we would need to write the value 0xC4 into the payload text file’s 3005th. On the for loop’s 3004th iteration, “i” would have a value of 0xBBC. Overwriting the MSB changes it to 0xBC4 which equals to 3012 in decimal causing the counter to skip past the memory locations containing the SFP and the counter variable “i” and directly into overwriting the RET address.

**Sploit 1 fix**

A simple way to fix this flaw in the programming is to first read the file and check the number of characters. If the number of characters read exceeds the buffer size, then the function would return and the buffer does not get overflowed.

**Sploit 2**

Environment variable stored at **0xffbfdf80**

Usage function has a buffer of size 200. Snprintf is used which means that we can’t overflow buffer past 200 characters. The parameter passed in is argv[0].

Hence we need to overwrite return address (Saved EIP) at address **ffbfdcdc.**

Since %n overwrites 32 bits regardless, we shall separate 0xffbfdcfc to upper and lower addresses. Therefore we overwrite only the top 2 bytes and lower 2 bytes of the memory content.

Print the stack to see where printf pointer is located at. The printf pointer will be pointing at address of the format string (argv[0]).

Format string will contain the address of the EIP which we need to overwrite.

Overwrite upper 2 bytes and lower 2 bytes (easier).

./s2 " BBBB%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x"

Usage: BBBB000000c8.08049860.ffbfdf0a.00000000.00000000.400197bc.ffbfdbe0.67617355.20203a65.42424242.78383025 backup|restore|ls pathname

The above prints out 11 blocks of memory. It is imperative that we use any random 1 byte character before “BBBB” to align BBBB to one memory block. Try it out yourself