Buffer Overflow Attack

Student ID _____
Student Name _____
Start Time _____
Finish Time _____

Note:ss
This lab cannot work on Code:Blocks

In this problem, you will mount a buffer overflow attack on your own program. As stated earlier, we do not condone using this or any other form of attack to gain unauthorized access to a system, but by doing this exercise, you will learn a lot about machine-level programming. Download the file bufoverflow.c and compile it to create an executable program.

In bufoverflow.c, you will find the following functions:

```
1 int getbuf()
2 {
3     char buf[32];
4     getxs(buf);
5     return 1;
6 }
7
8 void test()
9 {
10     int val;
11    printf("Type Hex string:");
12    val = getbuf();
13    printf("getbuf returned 0x%x\n", val);
14 }
```

The function getxs (also in bufoverflow.c) is similar to the library gets, except that it reads characters encoded as pairs of hex digits. For example, to give it a string "0123", the user would type in the string "30 31 32 33". The function ignores blank characters. Recall that decimal digit x has ASCII representation 0x3x. A typical execution of the program is as follows:

```
> bufoverflow
Type Hex string: 30 31 32 33
getbuf returned 0x1
```

Looking at the code for the getbuf function, it seems quite apparent that it will return value 1 whenever it is called. It appears as if the call to getxs has no effect. Your task is to make getbuf return 0xdeadbeef to test, simply by typing an appropriate hexadecimal string to the prompt.

Here are some ideas that will help you solve the problem:

- Use VC 6.0 debugger to create a disassembled version of buffer overflow. Study this closely to determine how the stack frame for getbuf is organized and how overflowing the buffer will alter the saved program state.
- Set a breakpoint within getbuf and run to this break point. Determine such parameters as the value of ebp and the saved value of any state that will be overwritten when you overflow the buffer.
- Determining the byte encoding of instruction sequences by hand is tedious and prone to errors. You can let tools do all of the work by writing an assembly code file containing the instructions and data you want to put on the stack. Assemble this file with disassembler.
- Keep in mind that your attack is very machine and compiler specific. You may need to alter your string when running on a different machine or with a different version of compiler.

Submission:

- Submit a report to show how you attack the program and show the input message and the screen shot of running result.