Extra credit: Buffer Overflow Vulnerability
(2 additional points to the final grade)

Firm Deadline: October 9th, 2006 by 11:59 pm. No extension will be granted.

1 Buffer overflow vulnerability

In this problem, you will attack buffer overflow vulnerability of the following program:

```
#include <stdio.h>
#include <unistd.h>
void buf_copy(char*);
int main(int argc, char ** argv)
{
        int retval;
        if (argc < 2)
        {
                fprintf(stderr, "usage: a.out message\n");
                exit(1);
        }
        buf_copy(argv[1]);
}
void buf_copy(char * buff)
{
        char buffer1[128];
        strcpy(buffer1, buff);
}
```

You will create an input string that once entered, calculates a factorial of the sum of the last two digits of your social security number. The C code for the factorial procedure is as follows:

```
int factorial(int num)
{
        int fact;
        if (num == 0) return 1; /* the termination condition */
        fact = factorial(num-1) * num; /* recursive call */
        return(fact);
}
```

The result of the factorial is to be stored in the variable *retval* in the main function. You will need to use a Linux box running X86 compatible processor (try `osage.unl.edu`). Follow the following hints to get start.

1. Create a sample C program that can utilize the factorial routine. Then compile the program using gcc -S flag. You will get a X86 assembly file in your directory (with .S extension). Examine the assembly instructions that perform factorial.

2. Create an assembly file or a mixture of C and assembly file that can calculate the factorial by itself (hardcode in the factorial value). You will need to readjust the assembly portion

of factorial so that it becomes the main function. The program appeared below illustrates how to mix assembly program with C program directly. More help can also be found from:

`http://www.ibiblio.org/gferg/ldp/GCC-Inline-Assembly-HOWTO.html#s3`:

```
#include <stdio.h>

int main(void)
{
    int x = 5, y = 6;
    __asm__ __volatile__("addl %%ebx,%%eax"
                         :"=a"(x)
                         :"a"(x), "b"(y)
                        );
    printf("x+y=%d\n", x);
    return 0;
}
```

3. Try debugging this new assembly program using GDB. GDB can decode each instruction into hexadecimal equivalence by using x/bx command.

4. Create a string of instructions that comprises of the hexadecimal equivalence. See if there are any NULL values (0x00) in your string. If there are, you must rewrite the instructions so that all NULL values are completely eliminated.

5. Test your string by hardcoding it into the test program. Check to see if the factorial result is written in *retval* variable.

6. If your factorial routine is executed, the next step is to take our hardcoded string and feed it into the vulnerable program. You also need to replace the return address on the stack so that your factorial routine is executed right after *buf_copy*. Refer to the following website for more information: http://www.infosecwriters.com/texts.php?op=display&id=134.

Some GDB commands you may need:
disassemble function
run
break function or line number
x/bx info frame
info args
info locals
backtrace