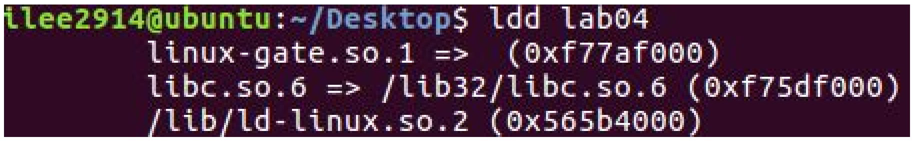
Ira Lee
Lab 4
10/27/17

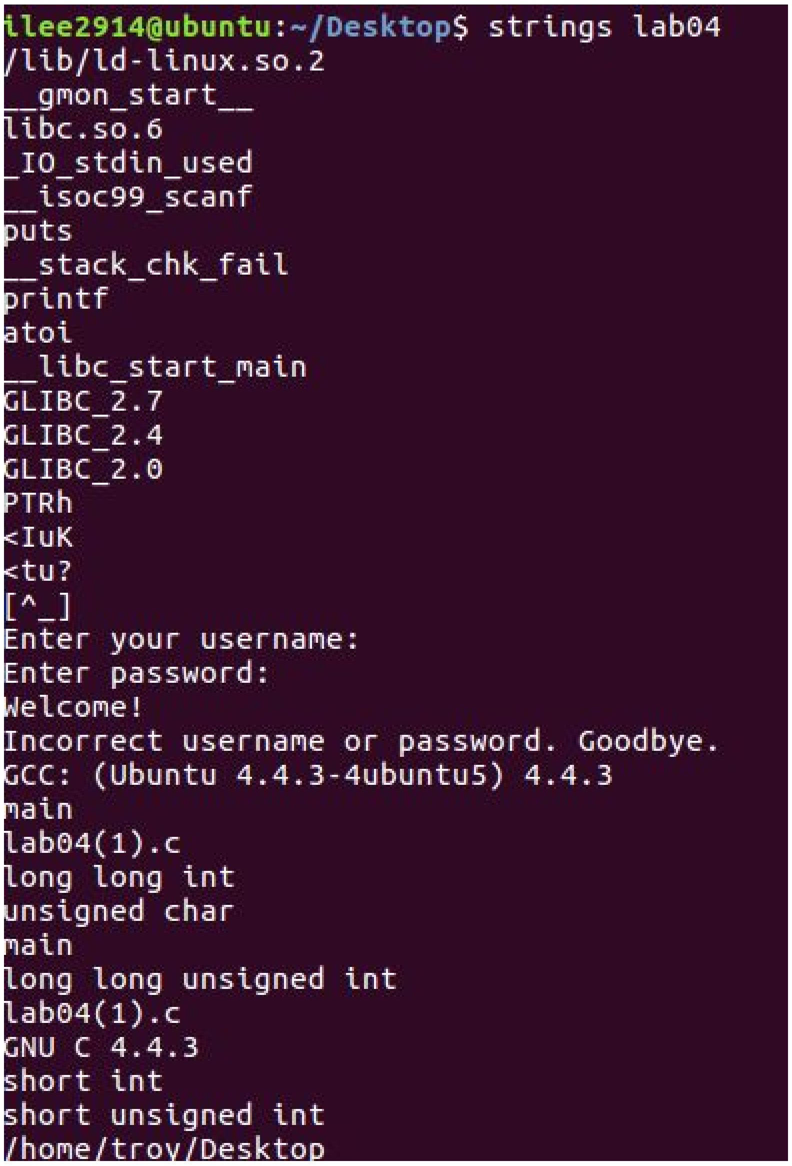
1a. ldd lab04



```
ilee2914@ubuntu:~/Desktop$ ldd lab04
        linux-gate.so.1 =>  (0xf77af000)
        libc.so.6 => /lib32/libc.so.6 (0xf75df000)
        /lib/ld-linux.so.2 (0x565b4000)
```

Shows the shared libraries.

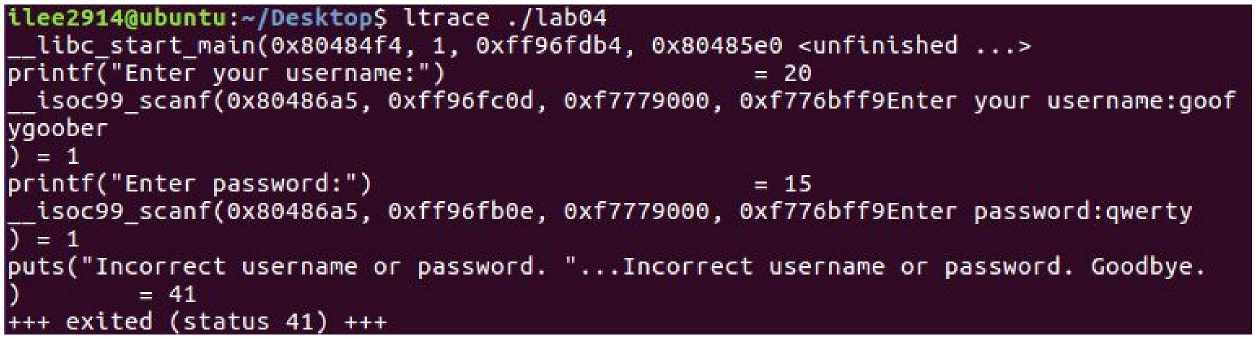
1b. strings lab04



```
ilee2914@ubuntu:~/Desktop$ strings lab04
/lib/ld-linux.so.2
__gmon_start__
libc.so.6
_IO_stdin_used
__isoc99_scanf
puts
__stack_chk_fail
printf
atoi
__libc_start_main
GLIBC_2.7
GLIBC_2.4
GLIBC_2.0
PTRh
<IuK
<tu?
[^_]
Enter your username:
Enter password:
Welcome!
Incorrect username or password. Goodbye.
GCC: (Ubuntu 4.4.3-4ubuntu5) 4.4.3
main
lab04(1).c
long long int
unsigned char
main
long long unsigned int
lab04(1).c
GNU C 4.4.3
short int
short unsigned int
/home/troy/Desktop
```

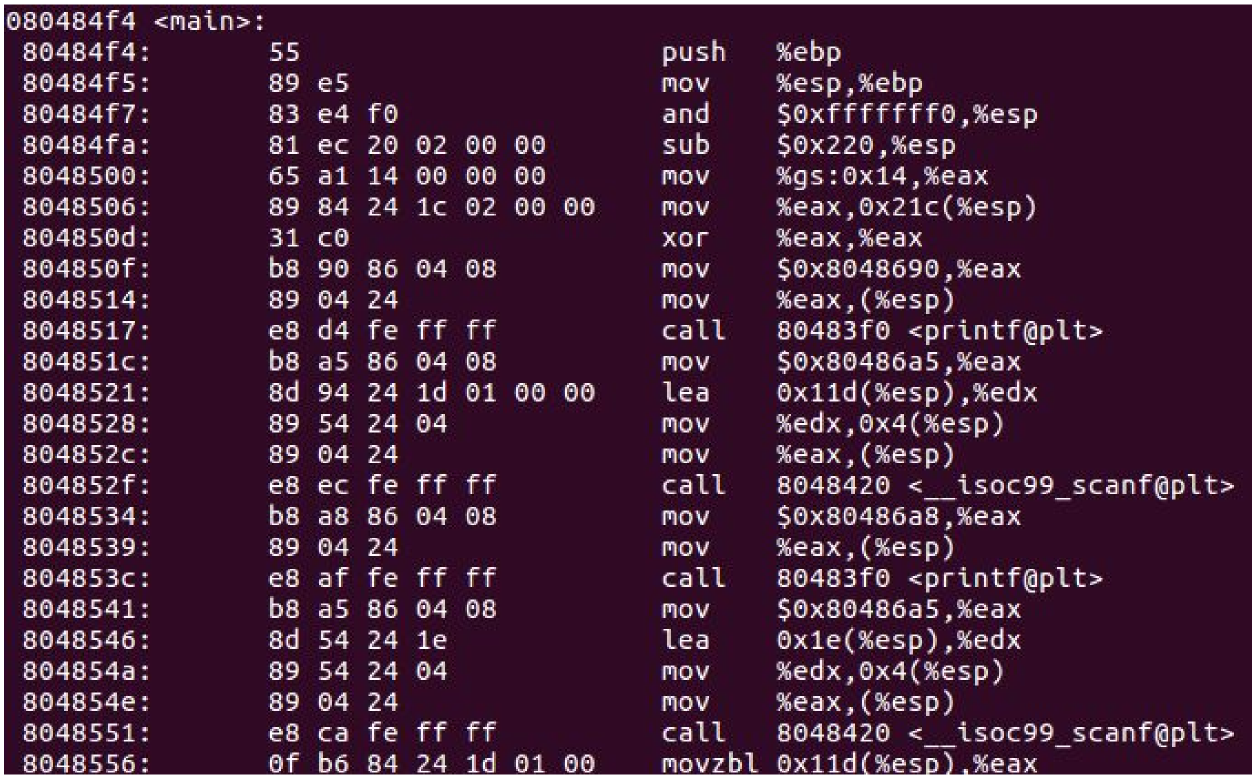
Shows all the printable characters used in the program. We can see here that the program is some sort of log in. It seems to also include some atoi calls, so I'm guessing either the username or password has some numbers in it.

1c. Ltrace lab04


```
ilee2914@ubuntu:~/Desktop$ ltrace ./lab04
__libc_start_main(0x80484f4, 1, 0xff96fdb4, 0x80485e0 <unfinished ...>
printf("Enter your username:")                   = 20
__isoc99_scanf(0x80486a5, 0xff96fc0d, 0xf7779000, 0xf776bff9Enter your username:goof
ygoober
) = 1
printf("Enter password:")                        = 15
__isoc99_scanf(0x80486a5, 0xff96fb0e, 0xf7779000, 0xf776bff9Enter password:qwerty
) = 1
puts("Incorrect username or password. "...Incorrect username or password. Goodbye.
)        = 41
+++ exited (status 41) +++
```

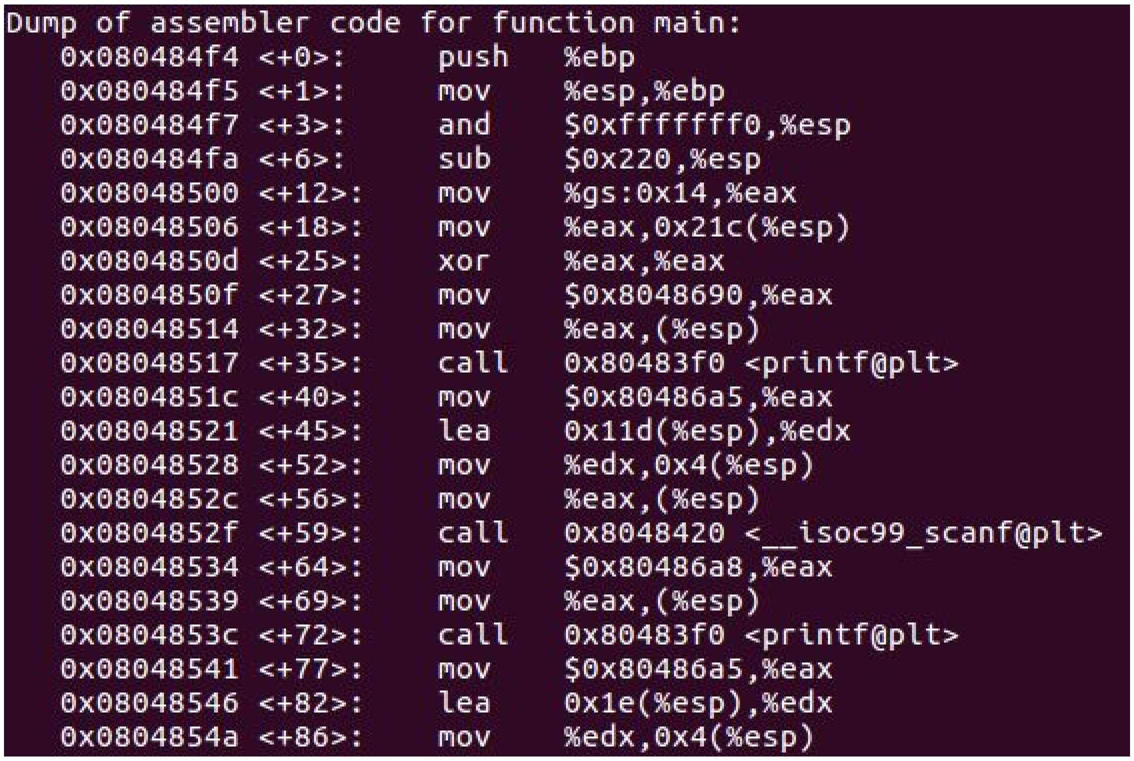
Runs the program and shows all the library and system calls

1d. objdump -d lab04


```
080484f4 <main>:
 80484f4:       55                      push   %ebp
 80484f5:       89 e5                   mov    %esp,%ebp
 80484f7:       83 e4 f0                and    $0xfffffff0,%esp
 80484fa:       81 ec 20 02 00 00       sub    $0x220,%esp
 8048500:       65 a1 14 00 00 00       mov    %gs:0x14,%eax
 8048506:       89 84 24 1c 02 00 00    mov    %eax,0x21c(%esp)
 804850d:       31 c0                   xor    %eax,%eax
 804850f:       b8 90 86 04 08          mov    $0x8048690,%eax
 8048514:       89 04 24                mov    %eax,(%esp)
 8048517:       e8 d4 fe ff ff          call   80483f0 <printf@plt>
 804851c:       b8 a5 86 04 08          mov    $0x80486a5,%eax
 8048521:       8d 94 24 1d 01 00 00    lea    0x11d(%esp),%edx
 8048528:       89 54 24 04             mov    %edx,0x4(%esp)
 804852c:       89 04 24                mov    %eax,(%esp)
 804852f:       e8 ec fe ff ff          call   8048420 <__isoc99_scanf@plt>
 8048534:       b8 a8 86 04 08          mov    $0x80486a8,%eax
 8048539:       89 04 24                mov    %eax,(%esp)
 804853c:       e8 af fe ff ff          call   80483f0 <printf@plt>
 8048541:       b8 a5 86 04 08          mov    $0x80486a5,%eax
 8048546:       8d 54 24 1e             lea    0x1e(%esp),%edx
 804854a:       89 54 24 04             mov    %edx,0x4(%esp)
 804854e:       89 04 24                mov    %eax,(%esp)
 8048551:       e8 ca fe ff ff          call   8048420 <__isoc99_scanf@plt>
 8048556:       0f b6 84 24 1d 01 00    movzbl 0x11d(%esp),%eax
```

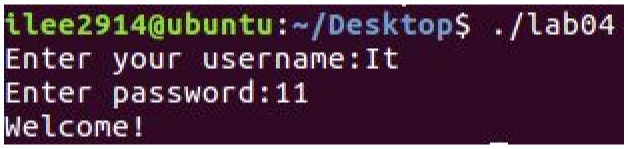
Shows the juicy commands and the hex code being called. From this we see that the jg command is on address 0x58f. We also see two compares on l and t.

1e. gdb diassem main



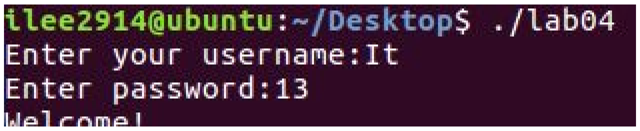
```
Dump of assembler code for function main:
   0x080484f4 <+0>:     push   %ebp
   0x080484f5 <+1>:     mov    %esp,%ebp
   0x080484f7 <+3>:     and    $0xfffffff0,%esp
   0x080484fa <+6>:     sub    $0x220,%esp
   0x08048500 <+12>:    mov    %gs:0x14,%eax
   0x08048506 <+18>:    mov    %eax,0x21c(%esp)
   0x0804850d <+25>:    xor    %eax,%eax
   0x0804850f <+27>:    mov    $0x8048690,%eax
   0x08048514 <+32>:    mov    %eax,(%esp)
   0x08048517 <+35>:    call   0x80483f0 <printf@plt>
   0x0804851c <+40>:    mov    $0x80486a5,%eax
   0x08048521 <+45>:    lea    0x11d(%esp),%edx
   0x08048528 <+52>:    mov    %edx,0x4(%esp)
   0x0804852c <+56>:    mov    %eax,(%esp)
   0x0804852f <+59>:    call   0x8048420 <__isoc99_scanf@plt>
   0x08048534 <+64>:    mov    $0x80486a8,%eax
   0x08048539 <+69>:    mov    %eax,(%esp)
   0x0804853c <+72>:    call   0x80483f0 <printf@plt>
   0x08048541 <+77>:    mov    $0x80486a5,%eax
   0x08048546 <+82>:    lea    0x1e(%esp),%edx
   0x0804854a <+86>:    mov    %edx,0x4(%esp)
```

Shows the assembler code of the calls. Same as objdump -d



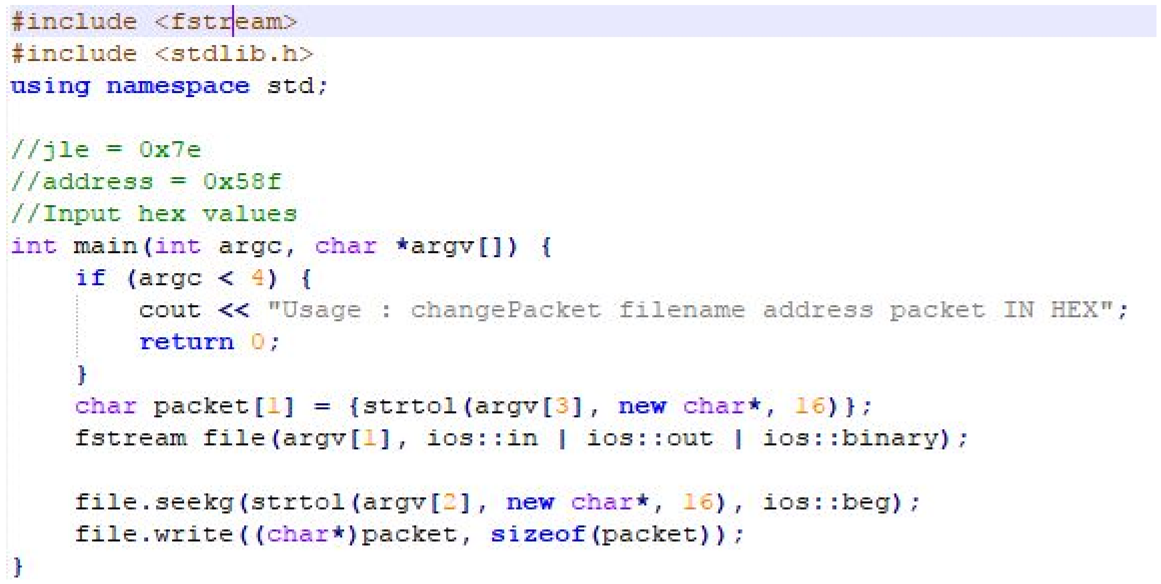
```
ilee2914@ubuntu:~/Desktop$ ./lab04
Enter your username:It
Enter password:11
Welcome!
```

Finally, from reading the code, it seems that the username is It from the first two cmp commands in the main function. We see a atoi call as well, so I assume the password is a number. A little after, it says cmpl 0xC, jg, which means to fail if the password is greater than 12. From what I tested, my hypothesis proved true.



```
ilee2914@ubuntu:~/Desktop$ ./lab04
Enter your username:It
Enter password:13
Welcome!
```

2. Code to change the password requirements



```
#include <fstream>
#include <stdlib.h>
using namespace std;

//jle = 0x7e
//address = 0x58f
//Input hex values
int main(int argc, char *argv[]) {
    if (argc < 4) {
        cout << "Usage : changePacket filename address packet IN HEX";
        return 0;
    }
    char packet[1] = {strtol(argv[3], new char*, 16)};
    fstream file(argv[1], ios::in | ios::out | ios::binary);

    file.seekg(strtol(argv[2], new char*, 16), ios::beg);
    file.write((char*)packet, sizeof(packet));
}
```

This code will allow the user to enter their own address and packet. Here we edit the the address 0x58F to become 0x7E, which is jl. With this, the condition is switched so that numbers less than 13 are not accepted, but numbers that are 13 or greater are accepted.

3. Can these changes be detected?
In an NTFS system, these changes can be easily detected because older versions of the binary file will exist. The time when the file was last modified will also be stored.