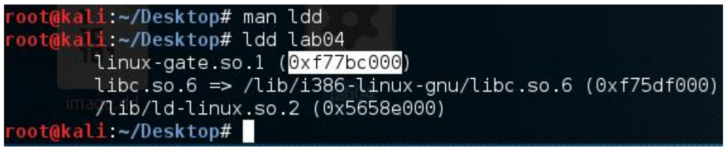
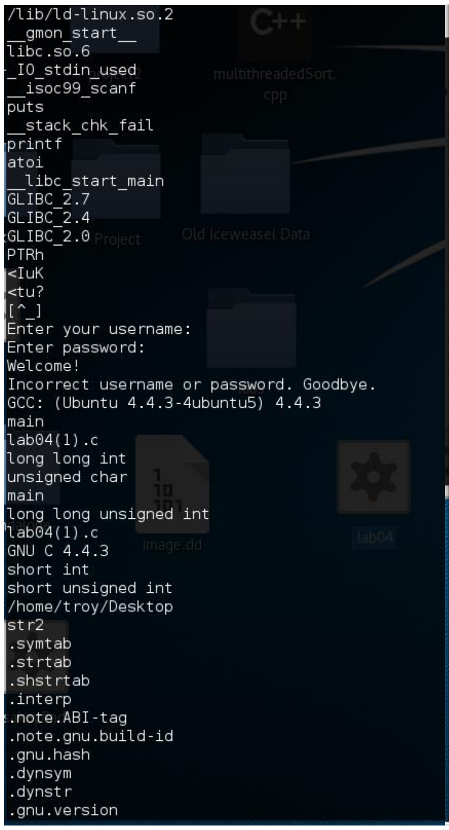
Christopher Mayol

1)

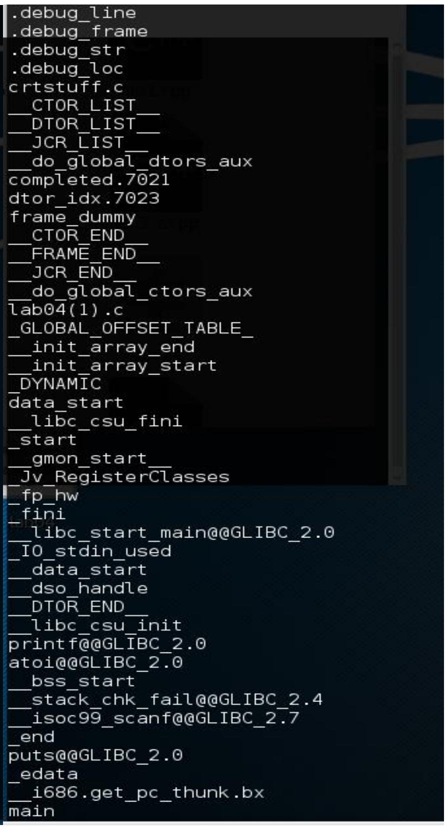


```
root@kali:~/Desktop# man ldd
root@kali:~/Desktop# ldd lab04
        linux-gate.so.1 (0xf77bc000)
        libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0xf75df000)
        /lib/ld-linux.so.2 (0x5658e000)
root@kali:~/Desktop#
```

Using the "ldd" command we see two shared libraries were found.



```
/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
str2
.symtab
.strtab
.shstrtab
.interp
.note.ABI-tag
.note.gnu.build-id
.gnu.hash
.dynsym
.dynstr
.gnu.version
```



```
.debug_line
.debug_frame
.debug_str
.debug_loc
crtstuff.c
__CTOR_LIST__
__DTOR_LIST__
__JCR_LIST__
__do_global_dtors_aux
completed.7021
dtor_idx.7023
frame_dummy
__CTOR_END__
__FRAME_END__
__JCR_END__
__do_global_ctors_aux
lab04(1).c
_GLOBAL_OFFSET_TABLE_
__init_array_end
__init_array_start
_DYNAMIC
data_start
__libc_csu_fini
_start
__gmon_start__
_Jv_RegisterClasses
_fp_hw
_fini
__libc_start_main@@GLIBC_2.0
_IO_stdin_used
__data_start
__dso_handle
__DTOR_END__
__libc_csu_init
printf@@GLIBC_2.0
atoi@@GLIBC_2.0
__bss_start
__stack_chk_fail@@GLIBC_2.4
__isoc99_scanf@@GLIBC_2.7
_end
puts@@GLIBC_2.0
_edata
__i686.get_pc_thunk.bx
main
```

Using the "strings" command we get the list of all the strings in the executable.

We see the the "printf","scanf","atoi" functions are being used and we see some interesting strings "Enter your username: ", "Enter password", "Welcome", "Incorrect username or password. Goodbye.".

From this we can kinda have a good idea what the program does.

Printf("Enter your username: ");

Scanf(user);

Printf("Enter password");

Scanf(pw);

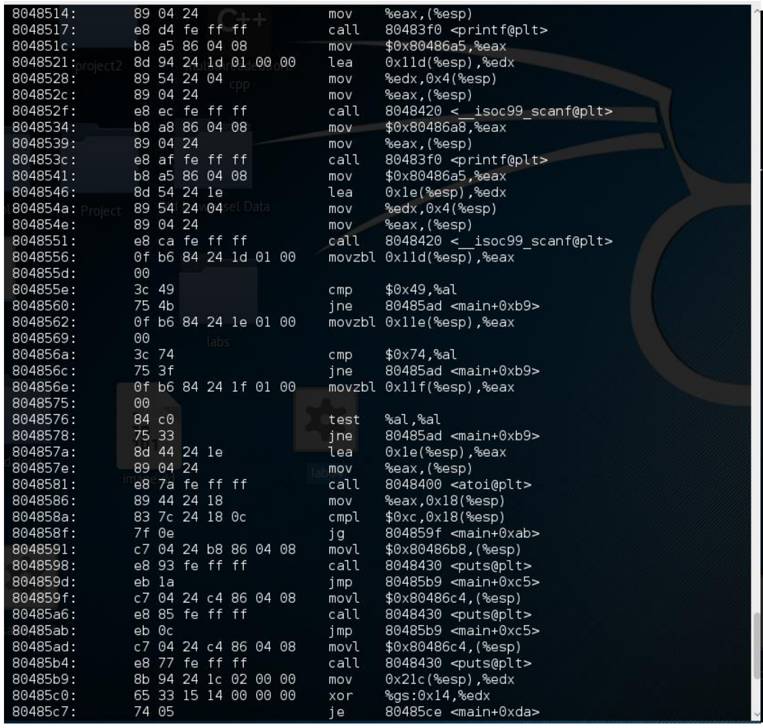
It uses atoi() function to convert a char to int. Then probably checks if the int matches with the hardcoded username or password.

    Printf("Welcome");

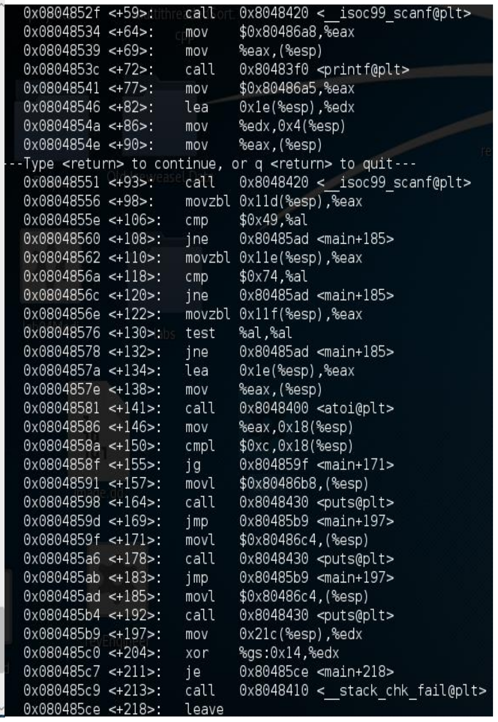
Else

    printf("Incorrect username or password. Goodbye.")

ltrace wasn't working for me



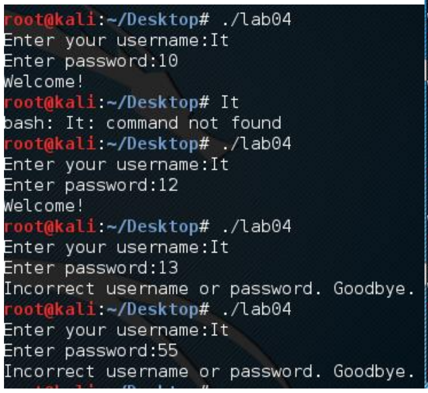
```
8048514:       89 04 24                mov    %eax,(%esp)
8048517:       e8 d4 fe ff ff          call   80483f0 <printf@plt>
804851c:       b8 a5 86 04 08          mov    $0x80486a5,%eax
8048521:       8d 94 24 1d 01 00 00    lea    0x11d(%esp),%edx
8048528:       89 54 24 04             mov    %edx,0x4(%esp)
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
804855d:       00
804855e:       3c 49                   cmp    $0x49,%al
8048560:       75 4b                   jne    80485ad <main+0xb9>
8048562:       0f b6 84 24 1e 01 00    movzbl 0x11e(%esp),%eax
8048569:       00
804856a:       3c 74                   cmp    $0x74,%al
804856c:       75 3f                   jne    80485ad <main+0xb9>
804856e:       0f b6 84 24 1f 01 00    movzbl 0x11f(%esp),%eax
8048575:       00
8048576:       84 c0                   test   %al,%al
8048578:       75 33                   jne    80485ad <main+0xb9>
804857a:       8d 44 24 1e             lea    0x1e(%esp),%eax
804857e:       89 04 24                mov    %eax,(%esp)
8048581:       e8 7a fe ff ff          call   8048400 <atoi@plt>
8048586:       89 44 24 18             mov    %eax,0x18(%esp)
804858a:       83 7c 24 18 0c          cmpl   $0xc,0x18(%esp)
804858f:       7f 0e                   jg     804859f <main+0xab>
8048591:       c7 04 24 b8 86 04 08    movl   $0x80486b8,(%esp)
8048598:       e8 93 fe ff ff          call   8048430 <puts@plt>
804859d:       eb 1a                   jmp    80485b9 <main+0xc5>
804859f:       c7 04 24 c4 86 04 08    movl   $0x80486c4,(%esp)
80485a6:       e8 85 fe ff ff          call   8048430 <puts@plt>
80485ab:       eb 0c                   jmp    80485b9 <main+0xc5>
80485ad:       c7 04 24 c4 86 04 08    movl   $0x80486c4,(%esp)
80485b4:       e8 77 fe ff ff          call   8048430 <puts@plt>
80485b9:       8b 94 24 1c 02 00 00    mov    0x21c(%esp),%edx
80485c0:       65 33 15 14 00 00 00    xor    %gs:0x14,%edx
80485c7:       74 05                   je     80485ce <main+0xda>
```



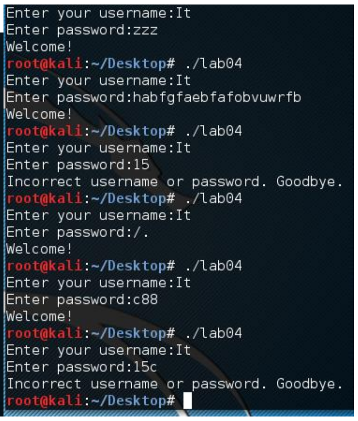
```
   0x0804852f <+59>:    call   0x8048420 <__isoc99_scanf@plt>
   0x08048534 <+64>:    mov    $0x80486a8,%eax
   0x08048539 <+69>:    mov    %eax,(%esp)
   0x0804853c <+72>:    call   0x80483f0 <printf@plt>
   0x08048541 <+77>:    mov    $0x80486a5,%eax
   0x08048546 <+82>:    lea    0x1e(%esp),%edx
   0x0804854a <+86>:    mov    %edx,0x4(%esp)
   0x0804854e <+90>:    mov    %eax,(%esp)
---Type <return> to continue, or q <return> to quit---
   0x08048551 <+93>:    call   0x8048420 <__isoc99_scanf@plt>
   0x08048556 <+98>:    movzbl 0x11d(%esp),%eax
   0x0804855e <+106>:   cmp    $0x49,%al
   0x08048560 <+108>:   jne    0x80485ad <main+185>
   0x08048562 <+110>:   movzbl 0x11e(%esp),%eax
   0x0804856a <+118>:   cmp    $0x74,%al
   0x0804856c <+120>:   jne    0x80485ad <main+185>
   0x0804856e <+122>:   movzbl 0x11f(%esp),%eax
   0x08048576 <+130>:   test   %al,%al
   0x08048578 <+132>:   jne    0x80485ad <main+185>
   0x0804857a <+134>:   lea    0x1e(%esp),%eax
   0x0804857e <+138>:   mov    %eax,(%esp)
   0x08048581 <+141>:   call   0x8048400 <atoi@plt>
   0x08048586 <+146>:   mov    %eax,0x18(%esp)
   0x0804858a <+150>:   cmpl   $0xc,0x18(%esp)
   0x0804858f <+155>:   jg     0x804859f <main+171>
   0x08048591 <+157>:   movl   $0x80486b8,(%esp)
   0x08048598 <+164>:   call   0x8048430 <puts@plt>
   0x0804859d <+169>:   jmp    0x80485b9 <main+197>
   0x0804859f <+171>:   movl   $0x80486c4,(%esp)
   0x080485a6 <+178>:   call   0x8048430 <puts@plt>
   0x080485ab <+183>:   jmp    0x80485b9 <main+197>
   0x080485ad <+185>:   movl   $0x80486c4,(%esp)
   0x080485b4 <+192>:   call   0x8048430 <puts@plt>
   0x080485b9 <+197>:   mov    0x21c(%esp),%edx
   0x080485c0 <+204>:   xor    %gs:0x14,%edx
   0x080485c7 <+211>:   je     0x80485ce <main+218>
   0x080485c9 <+213>:   call   0x8048410 <__stack_chk_fail@plt>
   0x080485ce <+218>:   leave
```

Using ObjDump and gdb commands which gave me more useful information. I was interested in the main function, I notice the printf and scanf functions were being called and after that in address 0x804855e we start comparing than jumping if the 0x49 ('I') is not in the al register, if they are equal continue to compare al with the next value 0x74 ('t'), else you will jump to the "wrong username or password" code. Testing this I found that the characters for the username is indeed 'I','t'.

Then I noticed the call to atoi () function, first mov eax to esp where eax is the password and copied to the stack, then the next instruction compares that value with 0xc, if the value is greater than 0xc it will jump probably to give you the error that the password does not match. So, basically we found the password, where the password can be any number less than or equal to 12 or any string that starts with a char because the atoi() function will then return a 0



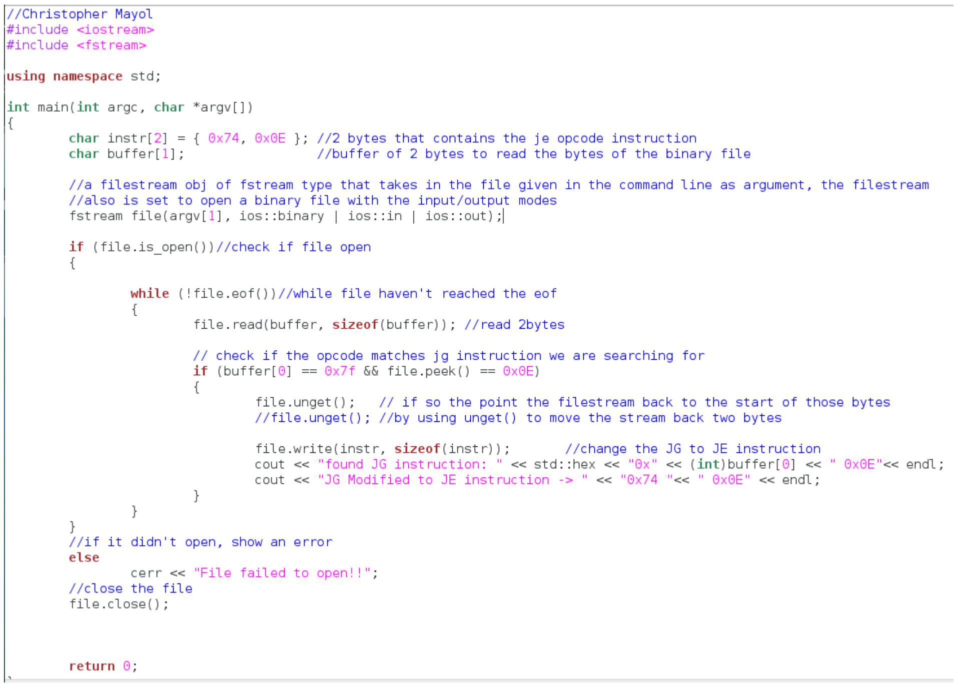
```
root@kali:~/Desktop# ./lab04
Enter your username:It
Enter password:10
Welcome!
root@kali:~/Desktop# It
bash: It: command not found
root@kali:~/Desktop# ./lab04
Enter your username:It
Enter password:12
Welcome!
root@kali:~/Desktop# ./lab04
Enter your username:It
Enter password:13
Incorrect username or password. Goodbye.
root@kali:~/Desktop# ./lab04
Enter your username:It
Enter password:55
Incorrect username or password. Goodbye.
```



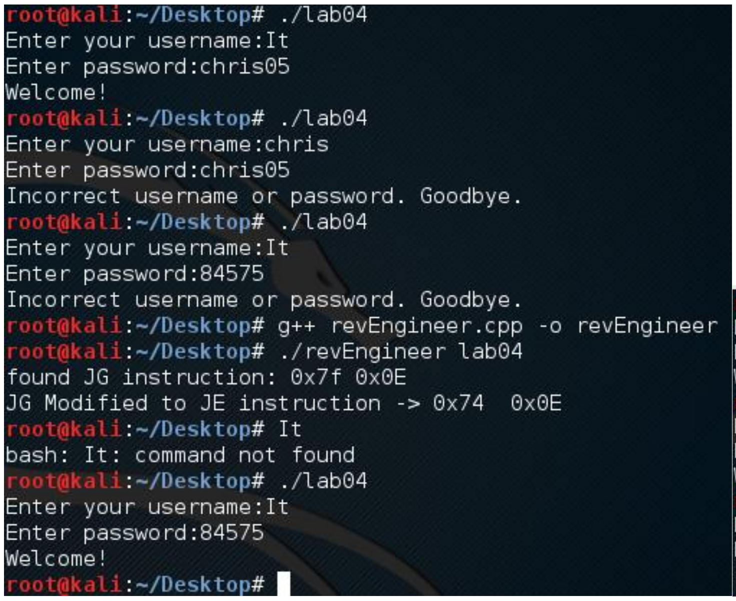
```
Enter your username:It
Enter password:zzz
Welcome!
root@kali:~/Desktop# ./lab04
Enter your username:It
Enter password:habfgfaebfafobvuwrfb
Welcome!
root@kali:~/Desktop# ./lab04
Enter your username:It
Enter password:15
Incorrect username or password. Goodbye.
root@kali:~/Desktop# ./lab04
Enter your username:It
Enter password:/.
Welcome!
root@kali:~/Desktop# ./lab04
Enter your username:It
Enter password:c88
Welcome!
root@kali:~/Desktop# ./lab04
Enter your username:It
Enter password:15c
Incorrect username or password. Goodbye.
root@kali:~/Desktop#
```

2)

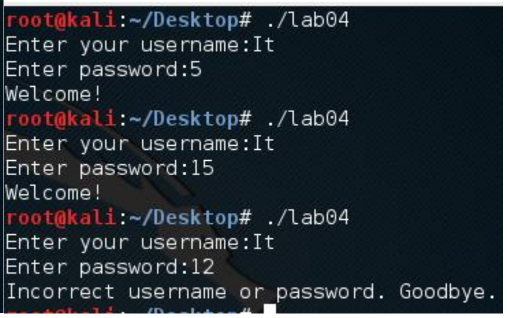
Program that finds the jump if greater instruction and modifies it to Jump if equals to



```cpp
//Christopher Mayol
#include <iostream>
#include <fstream>

using namespace std;

int main(int argc, char *argv[])
{
	char instr[2] = { 0x74, 0x0E }; //2 bytes that contains the je opcode instruction
	char buffer[1];                 //buffer of 2 bytes to read the bytes of the binary file

	//a filestream obj of fstream type that takes in the file given in the command line as argument, the filestream
	//also is set to open a binary file with the input/output modes
	fstream file(argv[1], ios::binary | ios::in | ios::out);

	if (file.is_open())//check if file open
	{

		while (!file.eof())//while file haven't reached the eof
		{
			file.read(buffer, sizeof(buffer)); //read 2bytes

			// check if the opcode matches jg instruction we are searching for
			if (buffer[0] == 0x7f && file.peek() == 0x0E)
			{
				file.unget();   // if so the point the filestream back to the start of those bytes
				//file.unget(); //by using unget() to move the stream back two bytes

				file.write(instr, sizeof(instr));       //change the JG to JE instruction
				cout << "found JG instruction: " << std::hex << "0x" << (int)buffer[0] << " 0x0E"<< endl;
				cout << "JG Modified to JE instruction -> " << "0x74 "<< " 0x0E" << endl;
			}
		}
	}
	//if it didn't open, show an error
	else
		cerr << "File failed to open!!";
	//close the file
	file.close();



	return 0;
```



```
root@kali:~/Desktop# ./lab04
Enter your username:It
Enter password:chris05
Welcome!
root@kali:~/Desktop# ./lab04
Enter your username:chris
Enter password:chris05
Incorrect username or password. Goodbye.
root@kali:~/Desktop# ./lab04
Enter your username:It
Enter password:84575
Incorrect username or password. Goodbye.
root@kali:~/Desktop# g++ revEngineer.cpp -o revEngineer
root@kali:~/Desktop# ./revEngineer lab04
found JG instruction: 0x7f 0x0E
JG Modified to JE instruction -> 0x74  0x0E
root@kali:~/Desktop# It
bash: It: command not found
root@kali:~/Desktop# ./lab04
Enter your username:It
Enter password:84575
Welcome!
root@kali:~/Desktop# 
```



```
root@kali:~/Desktop# ./lab04
Enter your username:It
Enter password:5
Welcome!
root@kali:~/Desktop# ./lab04
Enter your username:It
Enter password:15
Welcome!
root@kali:~/Desktop# ./lab04
Enter your username:It
Enter password:12
Incorrect username or password. Goodbye.
```

We see that before the patch passwords that are greater than 12 give an incorrect password error message. After the patch all the passwords that are not equal to 12 are correct. The screenshot in the right shows that after the patch the program accepts any password except the 12 or strings that start with 12. It makes sense because if we recall the program compares the password with the value 0xc (12) then it use to jump if greater than to the error message but we replaced "jg" with a jump if its equals to, so now if the password is 12 it will jump to the incorrect message.

3)

Is it possible to detect such a modification to a binary file on disk? If so, how?

Yes, if you have the source code or the original executable file, you can just compare both of the binary files with a hex editor and see the changes.