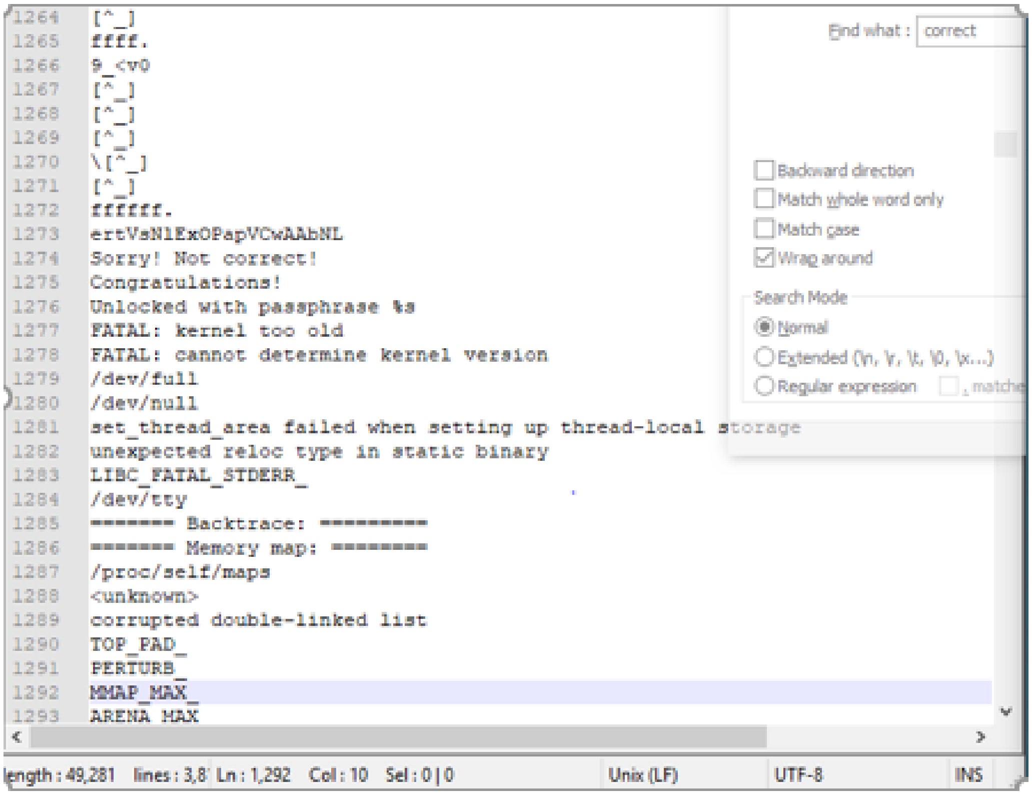
Joseph Weiss

Jbw40

Executable 1

Passcode= “ertVsNlExOPapVCwAAbNL”

I used my mystrings program to make a text file of my jbw40_1 executable (by typing **./mystrings jbw40_1>jbw40_1_txt** into thoth). Then I copied my **jbw40_1_txt** file into my private directory and opened it on windows in Notepad++ by using WinSCP. I then scrolled through the text file and found the location of the strings which tell the user “Sorry! Not correct!”, and “Congratulations!” Since these are the win and lose cases, I decided to start coping nearly strings to see if they were the ones the program was comparing the entered string to. I input the string “ertVsNlExOPapVCwAAbNL” right above the “not correct” response into the jbw40_1 executable and after misspelling it a few times got it correct. I tried inputting a few of the surrounding strings, to confirm the one I entered was the only answer, and they were all incorrect.



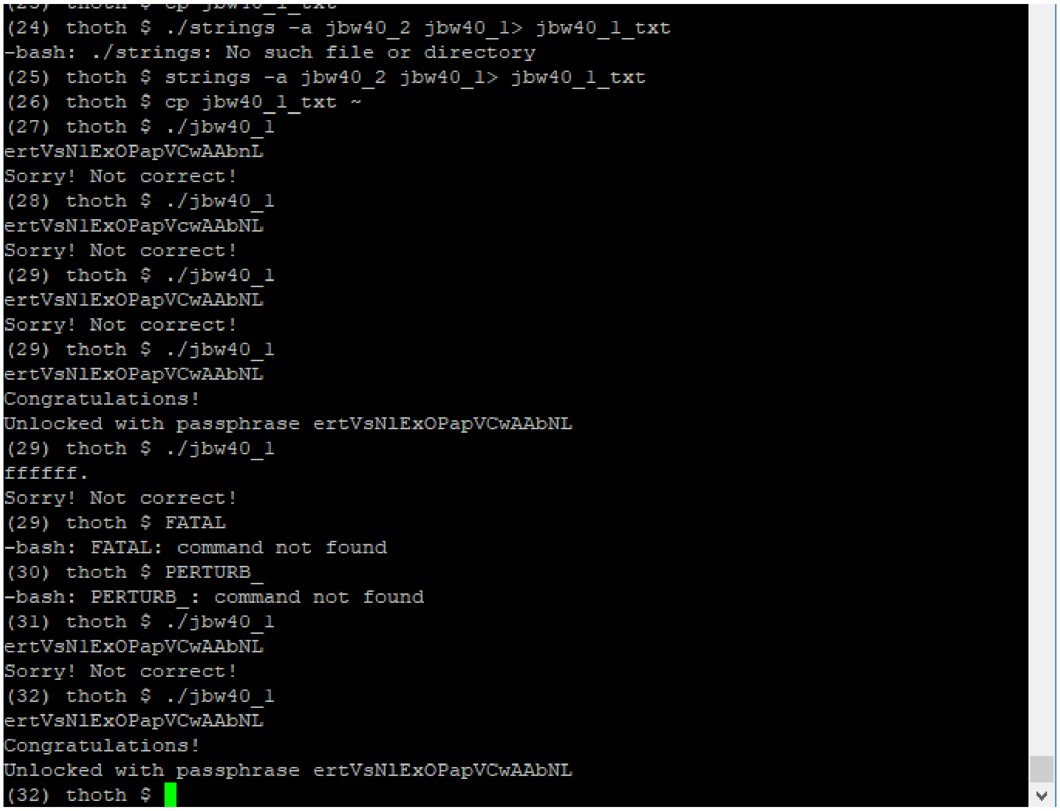
```
[^_]
ffff.
9_<v0
[^_]
[^_]
[^_]
\[^_]
[^_]
ffffff.
ertVsNlExOPapVCwAAbNL
Sorry! Not correct!
Congratulations!
Unlocked with passphrase %s
FATAL: kernel too old
FATAL: cannot determine kernel version
/dev/full
/dev/null
set_thread_area failed when setting up thread-local storage
unexpected reloc type in static binary
LIBC_FATAL_STDERR_
/dev/tty
======= Backtrace: =========
======= Memory map: ========
/proc/self/maps
<unknown>
corrupted double-linked list
TOP_PAD_
PERTURB_
MMAP_MAX_
ARENA_MAX
```

Find what : correct

□ Backward direction
□ Match whole word only
□ Match case
☑ Wrap around

Search Mode
● Normal
○ Extended (\n, \r, \t, \0, \x...)
○ Regular expression

length : 49,281 lines : 3,8 Ln : 1,292 Col : 10 Sel : 0 | 0 Unix (LF) UTF-8 INS

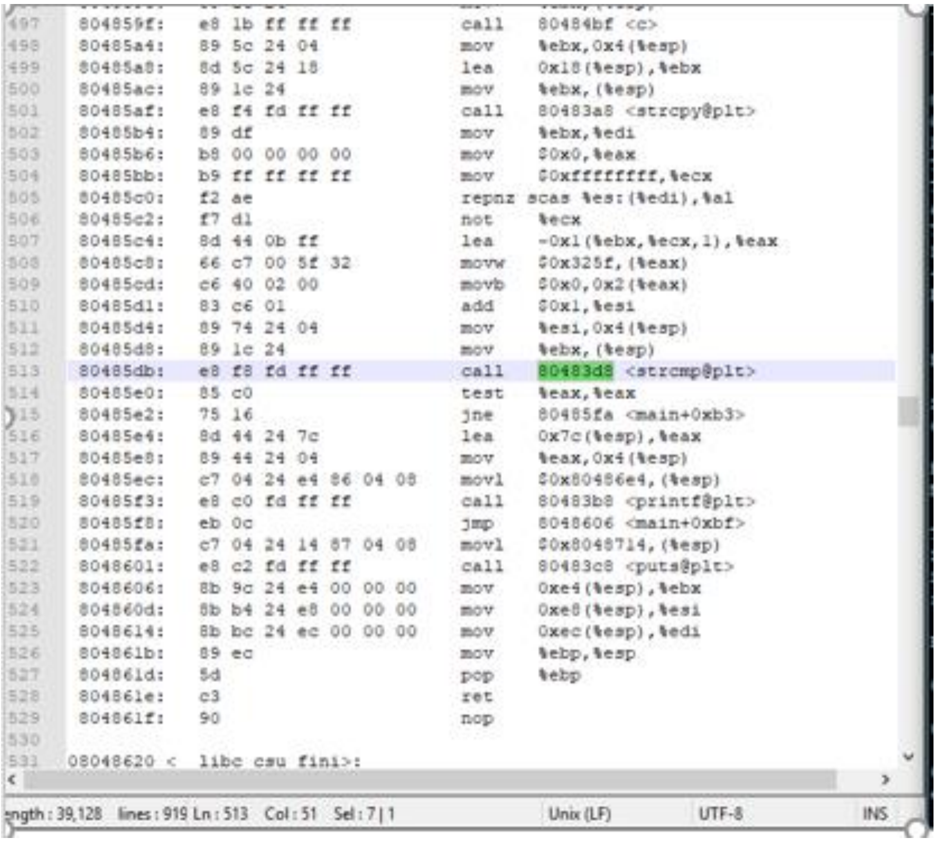


```
(23) thoth $ cp jbw40_1_txt
(24) thoth $ ./strings -a jbw40_2 jbw40_1> jbw40_1_txt
-bash: ./strings: No such file or directory
(25) thoth $ strings -a jbw40_2 jbw40_1> jbw40_1_txt
(26) thoth $ cp jbw40_1_txt ~
(27) thoth $ ./jbw40_1
ertVsNlExOPapVCwAAbnL
Sorry! Not correct!
(28) thoth $ ./jbw40_1
ertVsNlExOPapVcwAAbNL
Sorry! Not correct!
(29) thoth $ ./jbw40_1
ertVsNlExOPapVCwAAbNL
Sorry! Not correct!
(29) thoth $ ./jbw40_1
ertVsNlExOPapVCwAAbNL
Congratulations!
Unlocked with passphrase ertVsNlExOPapVCwAAbNL
(29) thoth $ ./jbw40_1
ffffff.
Sorry! Not correct!
(29) thoth $ FATAL
-bash: FATAL: command not found
(30) thoth $ PERTURB_
-bash: PERTURB_: command not found
(31) thoth $ ./jbw40_1
ertVsNlExOPapVCwAAbNL
Sorry! Not correct!
(32) thoth $ ./jbw40_1
ertVsNlExOPapVCwAAbNL
Congratulations!
Unlocked with passphrase ertVsNlExOPapVCwAAbNL
(32) thoth $
```

Executable 2

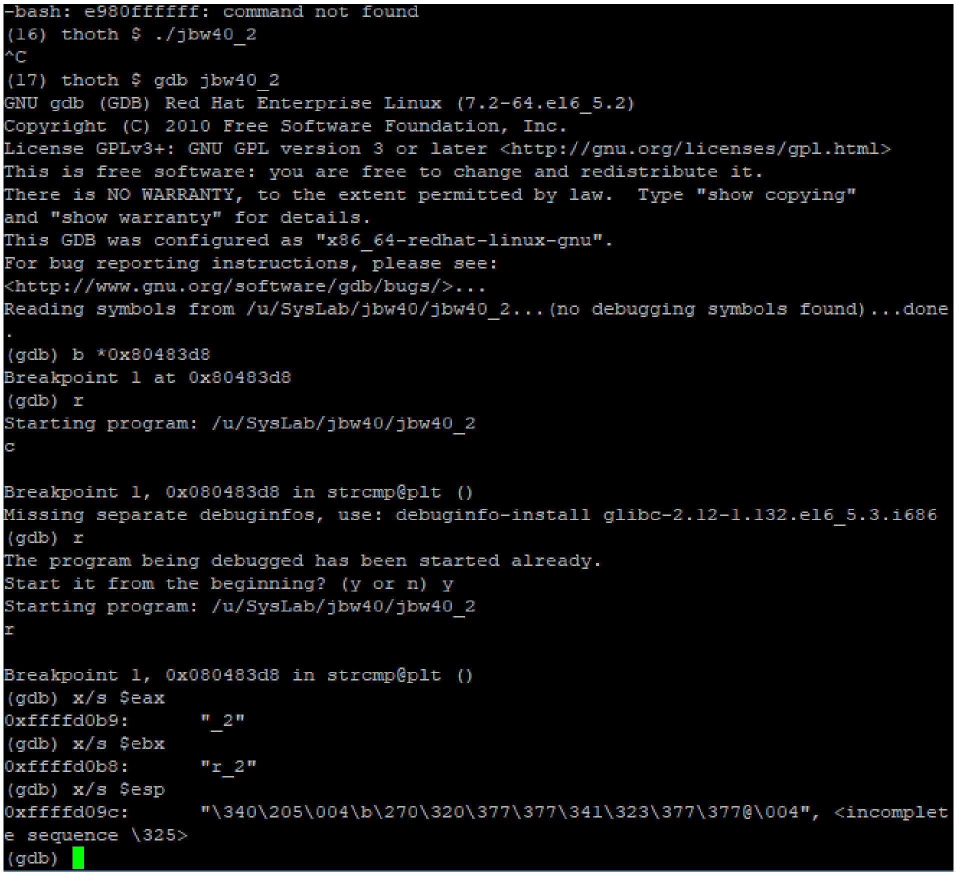
Passcode = "jbw40"

I started trying to crack the passcode to my jbw40_2 executable using the same means. I converted it into a text file, and found where "Congratulations!" was. There was no clear string that looked like the random passcode in the first executable, so I began typing in strings I found around it. However, none of these strings worked, so I decided against using the text file to find the passcode, since it was short and nothing looked like an obvious password. I then decided to try and find the answer similarly to how we did the Puzzle Lab in recitation. I used **objdump -D jbw40_2 > jbw40_2.dump** to get the assembly code for jbw40_2, and used WinSCP to open it in Notepad++. I looked through the calls made in main() and the most interesting was a call to strcmp, since it was likely that it was comparing something to whatever passcode I input. I opened the gdb debugger for jbw40_ 2, set a breakpoint to 0x80485db (where the strcmp is called) and ran the executable, using "r" as a test passcode. Then, once the program reached the breakpoint, I used **x/s $ebx** and **x/s %esp** to get the values of ebx and esp because in the move used right before the call to string compare, the value of esp is moved to ebx. I also tried **x/s $eax** because there is a test of eax after the call to strcmp.



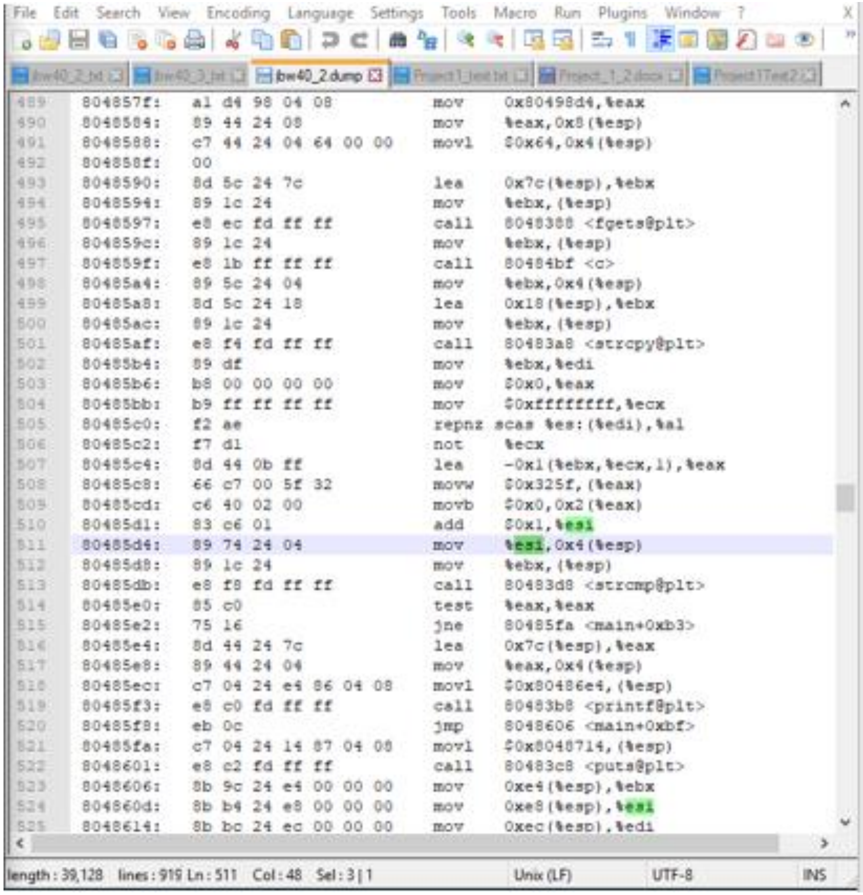
```
 804859f:	e8 1b ff ff ff       	call   80484bf <c>
 80485a4:	89 5c 24 04          	mov    %ebx,0x4(%esp)
 80485a8:	8d 5c 24 18          	lea    0x18(%esp),%ebx
 80485ac:	89 1c 24             	mov    %ebx,(%esp)
 80485af:	e8 f4 fd ff ff       	call   80483a8 <strcpy@plt>
 80485b4:	89 df                	mov    %ebx,%edi
 80485b6:	b8 00 00 00 00       	mov    $0x0,%eax
 80485bb:	b9 ff ff ff ff       	mov    $0xffffffff,%ecx
 80485c0:	f2 ae                	repnz scas %es:(%edi),%al
 80485c2:	f7 d1                	not    %ecx
 80485c4:	8d 44 0b ff          	lea    -0x1(%ebx,%ecx,1),%eax
 80485c8:	66 c7 00 5f 32       	movw   $0x325f,(%eax)
 80485cd:	c6 40 02 00          	movb   $0x0,0x2(%eax)
 80485d1:	83 c6 01             	add    $0x1,%esi
 80485d4:	89 74 24 04          	mov    %esi,0x4(%esp)
 80485d8:	89 1c 24             	mov    %ebx,(%esp)
 80485db:	e8 f8 fd ff ff       	call   80483d8 <strcmp@plt>
 80485e0:	85 c0                	test   %eax,%eax
 80485e2:	75 16                	jne    80485fa <main+0xb3>
 80485e4:	8d 44 24 7c          	lea    0x7c(%esp),%eax
 80485e8:	89 44 24 04          	mov    %eax,0x4(%esp)
 80485ec:	c7 04 24 e4 86 04 08 	movl   $0x80486e4,(%esp)
 80485f3:	e8 c0 fd ff ff       	call   80483b8 <printf@plt>
 80485f8:	eb 0c                	jmp    8048606 <main+0xbf>
 80485fa:	c7 04 24 14 87 04 08 	movl   $0x8048714,(%esp)
 8048601:	e8 c2 fd ff ff       	call   80483c8 <puts@plt>
 8048606:	8b 9c 24 e4 00 00 00 	mov    0xe4(%esp),%ebx
 804860d:	8b b4 24 e8 00 00 00 	mov    0xe8(%esp),%esi
 8048614:	8b bc 24 ec 00 00 00 	mov    0xec(%esp),%edi
 804861b:	89 ec                	mov    %ebp,%esp
 804861d:	5d                   	pop    %ebp
 804861e:	c3                   	ret    
 804861f:	90                   	nop

08048620 <  libc csu fini>:
```

ngth : 39,128   lines : 919 Ln : 513   Col : 51   Sel : 7 | 1
Unix (LF)
UTF-8
INS



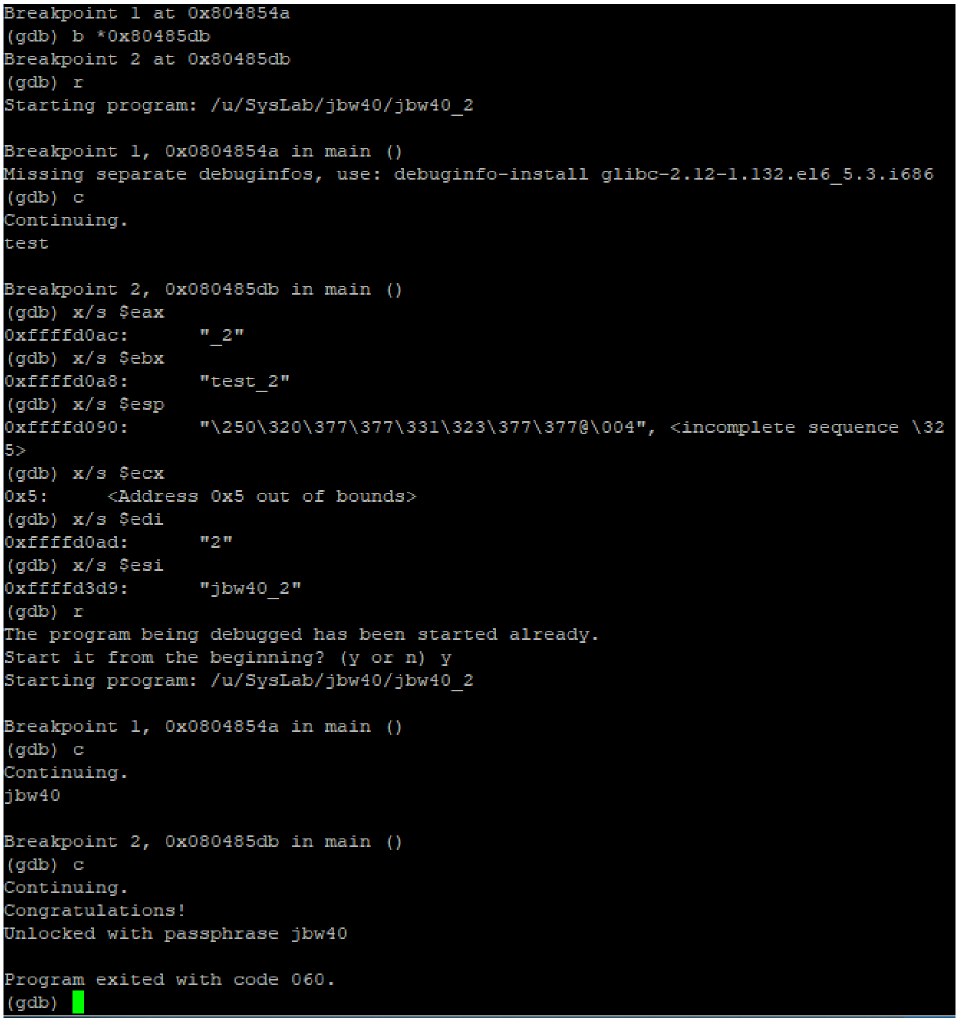
```
-bash: e980ffffff: command not found
(16) thoth $ ./jbw40_2
^C
(17) thoth $ gdb jbw40_2
GNU gdb (GDB) Red Hat Enterprise Linux (7.2-64.el6_5.2)
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /u/SysLab/jbw40/jbw40_2...(no debugging symbols found)...done
.
(gdb) b *0x80483d8
Breakpoint 1 at 0x80483d8
(gdb) r
Starting program: /u/SysLab/jbw40/jbw40_2
c

Breakpoint 1, 0x080483d8 in strcmp@plt ()
Missing separate debuginfos, use: debuginfo-install glibc-2.12-1.132.el6_5.3.i686
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /u/SysLab/jbw40/jbw40_2
r

Breakpoint 1, 0x080483d8 in strcmp@plt ()
(gdb) x/s $eax
0xffffd0b9:      "_2"
(gdb) x/s $ebx
0xffffd0b8:      "r_2"
(gdb) x/s $esp
0xffffd09c:      "\340\205\004\b\270\320\377\377\341\323\377\377@\004", <incomplet
e sequence \325>
(gdb) 
```

Eax was equal to “_2”, and ebx was “r_2”, and esp was some weird value. So I value I was entering (“r”) was having “_2” appended to it. The program wasn’t comparing eax and ebx for the final solution, because that would mean the user entered no value, which I tried but that was incorrect. I decided to look at other register values which might have the value being compared to ebx (the value I have with “_2” appended to it). I decided to check esi because two addresses before the call to strcmp moves the value of 0x4(%esp) into esi. I thought it was worth checking because the move right before strcmp has the value of esp being moved into ebx. When I did **x/s $esi** during the breakpoint at the strcmp**,** the value of esi was “jbw40_2”. So if ebx is being compared to “jbw40_2”, and ebx is equal to whatever I enter and “_2” added to it, that must mean if I enter “jbw40”, the program would append “_2” and it would equal “jbw40_2” and would be correct. I tried it and it worked.



```
804857f:   a1 d4 98 04 08          mov    0x80498d4,%eax
8048584:   89 44 24 08             mov    %eax,0x8(%esp)
8048588:   c7 44 24 04 64 00 00    movl   $0x64,0x4(%esp)
804858f:   00
8048590:   8d 5c 24 7c             lea    0x7c(%esp),%ebx
8048594:   89 1c 24                mov    %ebx,(%esp)
8048597:   e8 ec fd ff ff          call   8048388 <fgets@plt>
804859c:   89 1c 24                mov    %ebx,(%esp)
804859f:   e8 1b ff ff ff          call   80484bf <c>
80485a4:   89 5c 24 04             mov    %ebx,0x4(%esp)
80485a8:   8d 5c 24 18             lea    0x18(%esp),%ebx
80485ac:   89 1c 24                mov    %ebx,(%esp)
80485af:   e8 f4 fd ff ff          call   80483a8 <strcpy@plt>
80485b4:   89 df                   mov    %ebx,%edi
80485b6:   b8 00 00 00 00          mov    $0x0,%eax
80485bb:   b9 ff ff ff ff          mov    $0xffffffff,%ecx
80485c0:   f2 ae                   repnz scas %es:(%edi),%al
80485c2:   f7 d1                   not    %ecx
80485c4:   8d 44 0b ff             lea    -0x1(%ebx,%ecx,1),%eax
80485c8:   66 c7 00 5f 32          movw   $0x325f,(%eax)
80485cd:   c6 40 02 00             movb   $0x0,0x2(%eax)
80485d1:   83 c6 01                add    $0x1,%esi
80485d4:   89 74 24 04             mov    %esi,0x4(%esp)
80485d8:   89 1c 24                mov    %ebx,(%esp)
80485db:   e8 f8 fd ff ff          call   80483d8 <strcmp@plt>
80485e0:   85 c0                   test   %eax,%eax
80485e2:   75 16                   jne    80485fa <main+0xb3>
80485e4:   8d 44 24 7c             lea    0x7c(%esp),%eax
80485e8:   89 44 24 04             mov    %eax,0x4(%esp)
80485ec:   c7 04 24 e4 86 04 08    movl   $0x80486e4,(%esp)
80485f3:   e8 c0 fd ff ff          call   80483b8 <printf@plt>
80485f8:   eb 0c                   jmp    8048606 <main+0xbf>
80485fa:   c7 04 24 14 87 04 08    movl   $0x8048714,(%esp)
8048601:   e8 c2 fd ff ff          call   80483c8 <puts@plt>
8048606:   8b 9c 24 e4 00 00 00    mov    0xe4(%esp),%ebx
804860d:   8b b4 24 e8 00 00 00    mov    0xe8(%esp),%esi
8048614:   8b bc 24 ec 00 00 00    mov    0xec(%esp),%edi
```

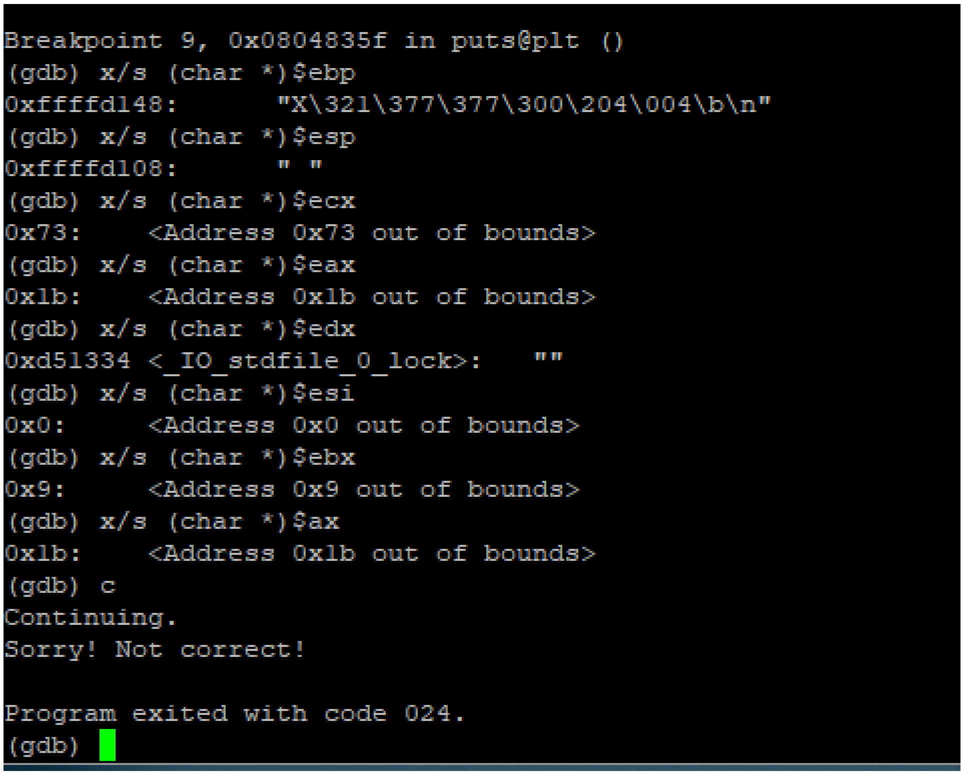
length : 39,128   lines : 919 Ln : 511   Col : 48   Sel : 3 | 1   Unix (LF)   UTF-8   INS



```
Breakpoint 1 at 0x804854a
(gdb) b *0x80485db
Breakpoint 2 at 0x80485db
(gdb) r
Starting program: /u/SysLab/jbw40/jbw40_2

Breakpoint 1, 0x0804854a in main ()
Missing separate debuginfos, use: debuginfo-install glibc-2.12-1.132.el6_5.3.i686
(gdb) c
Continuing.
test

Breakpoint 2, 0x080485db in main ()
(gdb) x/s $eax
0xffffd0ac:      "_2"
(gdb) x/s $ebx
0xffffd0a8:      "test_2"
(gdb) x/s $esp
0xffffd090:      "\250\320\377\377\331\323\377\377@\004", <incomplete sequence \32
5>
(gdb) x/s $ecx
0x5:     <Address 0x5 out of bounds>
(gdb) x/s $edi
0xffffd0ad:      "2"
(gdb) x/s $esi
0xffffd3d9:      "jbw40_2"
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /u/SysLab/jbw40/jbw40_2

Breakpoint 1, 0x0804854a in main ()
(gdb) c
Continuing.
jbw40

Breakpoint 2, 0x080485db in main ()
(gdb) c
Continuing.
Congratulations!
Unlocked with passphrase jbw40

Program exited with code 060.
(gdb) 
```

Executable 3

First I used my mystrings program to convert it into a text file, but it was short and no strings located around the “Congratulations” and “Sorry! Not correct!” strings which looked like they could be passwords. I then disassembled the executable using objdump, and since there was no main section, I began looking through the .text section. I set breakpoints to all the calls and many of the moves of the section, and then stepped through the code, getting the values for registers ebp, esp, ecx, eax, edx, esi, ebx, and ax at every breakpoint. However, none the values gave anything conclusive, though I did discover through testing it that the program accepted 9 characters, and would keep accepting user entries until they reached 9 chars in total.



```
Breakpoint 9, 0x0804835f in puts@plt ()
(gdb) x/s (char *)$ebp
0xffffd148:      "X\321\377\377\300\204\004\b\n"
(gdb) x/s (char *)$esp
0xffffd108:      " "
(gdb) x/s (char *)$ecx
0x73:    <Address 0x73 out of bounds>
(gdb) x/s (char *)$eax
0x1b:    <Address 0x1b out of bounds>
(gdb) x/s (char *)$edx
0xd51334 <_IO_stdfile_0_lock>:   ""
(gdb) x/s (char *)$esi
0x0:     <Address 0x0 out of bounds>
(gdb) x/s (char *)$ebx
0x9:     <Address 0x9 out of bounds>
(gdb) x/s (char *)$ax
0x1b:    <Address 0x1b out of bounds>
(gdb) c
Continuing.
Sorry! Not correct!

Program exited with code 024.
(gdb) 
```

I placed the breakpoint to an address early in the objdump of the .text portion of the executable, used the “ni” command to try and understand the way the program ran, and compared the addresses in the gdb debugger to the objdump to see why and how much the program was looping. I also tried inserting various entries into the passcode and breaking them up to see if they would have any effect on the variable values, but I noticed no obvious changes.