Forbes Turley

Project 2 Part B

6/27/17

Problem 1:

There were two viable methods to solving this problem and I tried both of them. The most straightforward method was to simply use the mystrings program on the fot3\_1 executable. This program works by scanning 8 bits of the input binary code at a time, and if the value is a valid ACII character, it is appended to the end of a string being built. As soon as a non-valid character is found, the program prints out the resulting string if its length is at least 4. It then continues the search. This yielded a huge number of strings, so I searched for ones that related to the output of the program and found this block of strings:

KXpLARkvWIYIoPlAKGKEElTKvsBCI

Sorry! Not correct!

Congratulations!

The first string of this group was accepted by the program.

The alternate method of evaluating this program is to place a breakpoint at main, and then disassemble the output. Looking at the assembly code, there is a cmp. By placing a breakpoint slightly before this at 0x08048307, and then performing an x/s on $esi, the output is "KxpLARkvWIYIoPlAKGKEElTKvsBCI".

I examined esi because it is by convention the string base address register.

Problem 2:

First, I tried using mystrings on this program as in the prior program, and found no strings that looked promising. I began to solve this problem by placing a breakpoint at main, and then performing a disassemble on the code. I saw that a function call to function d was occurring at 0x804854a, so I placed a breakpoint at function d. I then disassembled function d, and placed a number of breakpoints at the entries to each function called within d, which included s, c, and f. Upon further examination of the code within d, I saw that there was a test of flags occurring at line 0x080485a0 immediately followed by a call and a je. Because of this, I put a breakpoint immediately before this line and once again examined $esi. I chose to examine esi because it is by convention the string base address register. The output of $esi was **3.141593**, which seemed unlikely to be purely coincidental. I therefore tested this as the input to function fot3\_2, and was rewarded with a success message.

Problem 3:

This problem was very different from the other two. During my initial attempt I placed a breakpoint at main, but realized that there was no main function. I used “info file” to find the entry point, then placed a breakpoint at the entry point and tried to disassemble the code. However, I was thwarted again because I could not disassemble this function.

My next step was to use objdump to get the complete set of instructions within the file. I then went through the code line by line to understand the logic of the program, and determined that the following take place:

9 characters are input from the user. A counter at -0xc(%ebp) keeps track of how many characters have been input. One this has occurred, the characters are converted to lowercase (if they are uppercase letters). The resulting ASCII value then has 0x41 (65) subtracted from it. Note that if the ASCII value is already < 0x41, this will cause the result of the subtraction to be a negative number. Next, a comparison occurs between the results of the previous subtraction and 0x18 (24). If the value is less than 24 (but not negative), a jump will be skipped.

If the jump is skipped, the value will then be used as a shift for 0x1. So, if the value is 20, the result of the shift is 0x100000. The result of this shift is then bit-wise anded with 0x1104111 and if the result is non-zero, a second counter is incremented.

In order for the password to be accepted, this second counter must be incremented 3 and only 3 times. If the string fulfills this, it will be accepted.

The ASCII characters that would be considered valid characters to increment the second counter are uppercase A-Y. The problem is that because the characters are first converted to lowercase, NO VALID CHARACTERS exist that can increment the second counter. Therefore, there exist no valid passwords.

To confirm this is the correct conclusion, I changed the subtraction immediate from 0x41to 0x21 by performing set \*(char\*) 0x080484aa = 0x21. This then allows ASCII characters of ! through 6 to be considered valid by the above test, and then be used as a shift and anded. As a result of this, 5 yields a valid shift amount (0x1 shifted by 20 = 0x100000, which then anded results in a non-negative number). Then, a valid answer would be something of the form 555111111, or anything with 3 valid numbers that allow for shifts and subtractions to pass respective tests. 555111111 does in fact result in a success message when the subtraction immediate is changed to 0x21.

In conclusion, **no valid solutions exist** due to the conversion of upper case characters to lower case character.

I have included below my full dissection of the logic of this code.

Changing the subtraction immediate:

Reading symbols from fot3\_3...(no debugging symbols found)...done.

(gdb) b \*0x080484a8

Breakpoint 1 at 0x80484a8

(gdb) set \*(char\*) 0x080484aa = 0x21

Cannot access memory at address 0x80484aa

(gdb) b \*0x080483a0

Breakpoint 2 at 0x80483a0

(gdb) c

The program is not being run.

(gdb) r

Starting program: /home/forbes/OneDrive/CS449/Project2/fot3\_3

Breakpoint 2, 0x080483a0 in ?? ()

(gdb) set \*(char\*) 0x080484aa = 0x21

(gdb) c

Continuing.

5551111111111

Breakpoint 1, 0x080484a8 in ?? ()

(gdb) c

Continuing.

…. mutiple continues later

Continuing.

Congratulations!

Unlocked with passphrase 5551111111

[Inferior 1 (process 4049) exited with code 065

fot3\_3: file format elf32-i386

Disassembly of section .init:

080482f8 <.init>:

80482f8: 55 push %ebp

80482f9: 89 e5 mov %esp,%ebp

80482fb: 53 push %ebx

80482fc: 83 ec 04 sub $0x4,%esp

80482ff: e8 00 00 00 00 call 8048304 <getchar@plt-0x34>

8048304: 5b pop %ebx

8048305: 81 c3 d0 14 00 00 add $0x14d0,%ebx

804830b: 8b 93 fc ff ff ff mov -0x4(%ebx),%edx

8048311: 85 d2 test %edx,%edx

8048313: 74 05 je 804831a <getchar@plt-0x1e>

8048315: e8 2e 00 00 00 call 8048348 <\_\_gmon\_start\_\_@plt>

804831a: e8 11 01 00 00 call 8048430 <tolower@plt+0xa8>

804831f: e8 8c 02 00 00 call 80485b0 <tolower@plt+0x228>

8048324: 58 pop %eax

8048325: 5b pop %ebx

8048326: c9 leave

8048327: c3 ret

Disassembly of section .plt:

08048328 <getchar@plt-0x10>:

8048328: ff 35 d8 97 04 08 pushl 0x80497d8

804832e: ff 25 dc 97 04 08 jmp \*0x80497dc

8048334: 00 00 add %al,(%eax)

...

08048338 <getchar@plt>:

8048338: ff 25 e0 97 04 08 jmp \*0x80497e0

804833e: 68 00 00 00 00 push $0x0

8048343: e9 e0 ff ff ff jmp 8048328 <getchar@plt-0x10>

08048348 <\_\_gmon\_start\_\_@plt>:

8048348: ff 25 e4 97 04 08 jmp \*0x80497e4

804834e: 68 08 00 00 00 push $0x8

8048353: e9 d0 ff ff ff jmp 8048328 <getchar@plt-0x10>

08048358 <\_\_libc\_start\_main@plt>:

8048358: ff 25 e8 97 04 08 jmp \*0x80497e8

804835e: 68 10 00 00 00 push $0x10

8048363: e9 c0 ff ff ff jmp 8048328 <getchar@plt-0x10>

08048368 <printf@plt>:

8048368: ff 25 ec 97 04 08 jmp \*0x80497ec

804836e: 68 18 00 00 00 push $0x18

8048373: e9 b0 ff ff ff jmp 8048328 <getchar@plt-0x10>

08048378 <puts@plt>:

8048378: ff 25 f0 97 04 08 jmp \*0x80497f0

804837e: 68 20 00 00 00 push $0x20

8048383: e9 a0 ff ff ff jmp 8048328 <getchar@plt-0x10>

08048388 <tolower@plt>:

8048388: ff 25 f4 97 04 08 jmp \*0x80497f4

804838e: 68 28 00 00 00 push $0x28

8048393: e9 90 ff ff ff jmp 8048328 <getchar@plt-0x10>

Disassembly of section .text:

080483a0 <.text>:

80483a0: 31 ed xor %ebp,%ebp Setting up the stack

80483a2: 5e pop %esi

80483a3: 89 e1 mov %esp,%ecx

80483a5: 83 e4 f0 and $0xfffffff0,%esp Doing the logic discussed in class to make sure that stack frame begins on a multiple of 16

80483a8: 50 push %eax

80483a9: 54 push %esp

80483aa: 52 push %edx

80483ab: 68 40 85 04 08 push $0x8048540

80483b0: 68 50 85 04 08 push $0x8048550

80483b5: 51 push %ecx

80483b6: 56 push %esi

80483b7: 68 24 85 04 08 push $0x8048524

80483bc: e8 97 ff ff ff call 8048358 <\_\_libc\_start\_main@plt>

80483c1: f4 hlt

80483c2: 90 nop

80483c3: 90 nop

80483c4: 90 nop

80483c5: 90 nop

80483c6: 90 nop

80483c7: 90 nop

80483c8: 90 nop

80483c9: 90 nop

80483ca: 90 nop

80483cb: 90 nop

80483cc: 90 nop

80483cd: 90 nop

80483ce: 90 nop

80483cf: 90 nop

80483d0: 55 push %ebp

80483d1: 89 e5 mov %esp,%ebp

80483d3: 53 push %ebx

80483d4: 83 ec 04 sub $0x4,%esp

80483d7: 80 3d fc 97 04 08 00 cmpb $0x0,0x80497fc

80483de: 75 3f jne 804841f <tolower@plt+0x97>

80483e0: a1 00 98 04 08 mov 0x8049800,%eax

80483e5: bb 00 97 04 08 mov $0x8049700,%ebx

80483ea: 81 eb fc 96 04 08 sub $0x80496fc,%ebx

80483f0: c1 fb 02 sar $0x2,%ebx

80483f3: 83 eb 01 sub $0x1,%ebx

80483f6: 39 d8 cmp %ebx,%eax

80483f8: 73 1e jae 8048418 <tolower@plt+0x90>

80483fa: 8d b6 00 00 00 00 lea 0x0(%esi),%esi

8048400: 83 c0 01 add $0x1,%eax

8048403: a3 00 98 04 08 mov %eax,0x8049800

8048408: ff 14 85 fc 96 04 08 call \*0x80496fc(,%eax,4)

804840f: a1 00 98 04 08 mov 0x8049800,%eax

8048414: 39 d8 cmp %ebx,%eax

8048416: 72 e8 jb 8048400 <tolower@plt+0x78>

8048418: c6 05 fc 97 04 08 01 movb $0x1,0x80497fc

804841f: 83 c4 04 add $0x4,%esp

8048422: 5b pop %ebx

8048423: 5d pop %ebp

8048424: c3 ret

8048425: 8d 74 26 00 lea 0x0(%esi,%eiz,1),%esi

8048429: 8d bc 27 00 00 00 00 lea 0x0(%edi,%eiz,1),%edi

8048430: 55 push %ebp

8048431: 89 e5 mov %esp,%ebp

8048433: 83 ec 18 sub $0x18,%esp

8048436: a1 04 97 04 08 mov 0x8049704,%eax

804843b: 85 c0 test %eax,%eax

804843d: 74 12 je 8048451 <tolower@plt+0xc9>

804843f: b8 00 00 00 00 mov $0x0,%eax

8048444: 85 c0 test %eax,%eax

8048446: 74 09 je 8048451 <tolower@plt+0xc9>

8048448: c7 04 24 04 97 04 08 movl $0x8049704,(%esp)

804844f: ff d0 call \*%eax

8048451: c9 leave

8048452: c3 ret

8048453: 90 nop

8048454: 55 push %ebp

8048455: 89 e5 mov %esp,%ebp

8048457: 53 push %ebx

8048458: 83 ec 34 sub $0x34,%esp

804845b: c7 45 f0 00 00 00 00 movl $0x0,-0x10(%ebp) Counter1 init to 0

8048462: c7 45 f4 00 00 00 00 movl $0x0,-0xc(%ebp) Counter2 init to 0

8048469: eb 10 jmp 804847b <tolower@plt+0xf3> if counter2 is greater than 9 jump, otherwise do the following:

804846b: 8b 5d f4 mov -0xc(%ebp),%ebx

804846e: e8 c5 fe ff ff call 8048338 <getchar@plt> fetch a character

8048473: 88 44 1d e5 mov %al,-0x1b(%ebp,%ebx,1)

8048477: 83 45 f4 01 addl $0x1,-0xc(%ebp) Increment couter2

804847b: 83 7d f4 09 cmpl $0x9,-0xc(%ebp) Compare counter2 to 9 (It only looks at 9 characters)

804847f: 7e ea jle 804846b <tolower@plt+0xe3> This is a loop back to 804846b

8048481: 8b 45 f4 mov -0xc(%ebp),%eax

8048484: c6 44 05 e5 00 movb $0x0,-0x1b(%ebp,%eax,1)

8048489: c7 45 f4 01 00 00 00 movl $0x1,-0xc(%ebp) Reset counter2 to 1

8048490: eb 3d jmp 80484cf <tolower@plt+0x147>

8048492: 8b 45 f4 mov -0xc(%ebp),%eax

8048495: 83 e8 01 sub $0x1,%eax ` Subtract 1 from counter2

8048498: 0f b6 44 05 e5 movzbl -0x1b(%ebp,%eax,1),%eax

804849d: 0f be c0 movsbl %al,%eax

80484a0: 89 04 24 mov %eax,(%esp)

80484a3: e8 e0 fe ff ff call 8048388 <tolower@plt>

80484a8: 83 e8 41 sub $0x41,%eax Sub. 65 from current char

80484ab: 83 f8 18 cmp $0x18,%eax If UNSIGNED less than 24, ja will NOT execute. As a result ja ALWAYS executes.

80484ae: 77 1b ja 80484cb Jumps to 80484cb

80484b0: ba 01 00 00 00 mov $0x1,%edx Move 1 to edx

80484b5: 89 d3 mov %edx,%ebx Move edx to ebx

80484b7: 89 c1 mov %eax,%ecx Move eax to ecx

80484b9: d3 e3 shl %cl,%ebx Shifts left by value in eac (which is char - 65)

80484bb: 89 d8 mov %ebx,%eax

80484bd: 25 11 41 10 01 and $0x1104111,%eax ANDS value from shift with this. If resulting value is non-zero, the counter1 is incremented.

80484c2: 85 c0 test %eax,%eax This jumps to tolower if eax is 0.

80484c4: 74 05 je 80484cb <tolower@plt+0x143>

80484c6: 83 45 f0 01 addl $0x1,-0x10(%ebp) Adds 1 to the counter2

80484ca: 90 nop

80484cb: 83 45 f4 01 addl $0x1,-0xc(%ebp) Adds 1 to counter2

80484cf: 83 7d f4 0a cmpl $0xa,-0xc(%ebp) Compares counter2 with 10

80484d3: 7e bd jle 8048492 <tolower@plt+0x10a> If counter 2 is less than or equal to 10, jump to 8048492

80484d5: 83 7d f0 03 cmpl $0x3,-0x10(%ebp) else, if counter1 equals 3

set \*(char\*) 0x080484aa = 0x21

2 = 20,000 NO GOOD

3 = 40,000 NO GOOD

4 = 80,000 NO GOOD

5 = 100,000 GOOD

6 - 200,000 NO GOOD

Somehow, the comparison above seems to be preventing me from getting to the accept string below.

80484d9: 75 16 jne 80484f1 <tolower@plt+0x169> Jump to

80484db: b8 04 86 04 08 mov $0x8048604,%eax This is the address of the ACCEPT string

80484e0: 8d 55 e5 lea -0x1b(%ebp),%edx

80484e3: 89 54 24 04 mov %edx,0x4(%esp)

80484e7: 89 04 24 mov %eax,(%esp)

80484ea: e8 79 fe ff ff call 8048368 <printf@plt>

80484ef: eb 0c jmp 80484fd <tolower@plt+0x175>

80484f1: c7 04 24 32 86 04 08 movl $0x8048632,(%esp) This is the address of the REJECT string.

80484f8: e8 7b fe ff ff call 8048378 <puts@plt>

80484fd: 83 c4 34 add $0x34,%esp

8048500: 5b pop %ebx

8048501: 5d pop %ebp

8048502: c3 ret

8048503: 55 push %ebp

8048504: 89 e5 mov %esp,%ebp

8048506: 83 ec 08 sub $0x8,%esp

8048509: e8 46 ff ff ff call 8048454 <tolower@plt+0xcc>

804850e: c9 leave

804850f: c3 ret

8048510: 55 push %ebp

8048511: 89 e5 mov %esp,%ebp

8048513: 83 ec 18 sub $0x18,%esp

8048516: c7 45 f4 46 86 04 08 movl $0x8048646,-0xc(%ebp)

804851d: e8 e1 ff ff ff call 8048503 <tolower@plt+0x17b>

8048522: c9 leave

8048523: c3 ret

8048524: 55 push %ebp

8048525: 89 e5 mov %esp,%ebp

8048527: 83 e4 f0 and $0xfffffff0,%esp

804852a: e8 e1 ff ff ff call 8048510 <tolower@plt+0x188>

804852f: 89 ec mov %ebp,%esp

8048531: 5d pop %ebp

8048532: c3 ret

8048533: 90 nop

8048534: 90 nop

8048535: 90 nop

8048536: 90 nop

8048537: 90 nop

8048538: 90 nop

8048539: 90 nop

804853a: 90 nop

804853b: 90 nop

804853c: 90 nop

804853d: 90 nop

804853e: 90 nop

804853f: 90 nop

8048540: 55 push %ebp

8048541: 89 e5 mov %esp,%ebp

8048543: 5d pop %ebp

8048544: c3 ret

8048545: 66 66 2e 0f 1f 84 00 data16 nopw %cs:0x0(%eax,%eax,1)

804854c: 00 00 00 00

8048550: 55 push %ebp

8048551: 89 e5 mov %esp,%ebp

8048553: 57 push %edi

8048554: 56 push %esi

8048555: 53 push %ebx

8048556: e8 4f 00 00 00 call 80485aa <tolower@plt+0x222>

804855b: 81 c3 79 12 00 00 add $0x1279,%ebx

8048561: 83 ec 1c sub $0x1c,%esp

8048564: e8 8f fd ff ff call 80482f8 <getchar@plt-0x40>

8048569: 8d bb 20 ff ff ff lea -0xe0(%ebx),%edi

804856f: 8d 83 20 ff ff ff lea -0xe0(%ebx),%eax

8048575: 29 c7 sub %eax,%edi

8048577: c1 ff 02 sar $0x2,%edi

804857a: 85 ff test %edi,%edi

804857c: 74 24 je 80485a2 <tolower@plt+0x21a>

804857e: 31 f6 xor %esi,%esi

8048580: 8b 45 10 mov 0x10(%ebp),%eax

8048583: 89 44 24 08 mov %eax,0x8(%esp)

8048587: 8b 45 0c mov 0xc(%ebp),%eax

804858a: 89 44 24 04 mov %eax,0x4(%esp)

804858e: 8b 45 08 mov 0x8(%ebp),%eax

8048591: 89 04 24 mov %eax,(%esp)

8048594: ff 94 b3 20 ff ff ff call \*-0xe0(%ebx,%esi,4)

804859b: 83 c6 01 add $0x1,%esi

804859e: 39 fe cmp %edi,%esi

80485a0: 72 de jb 8048580 <tolower@plt+0x1f8>

80485a2: 83 c4 1c add $0x1c,%esp

80485a5: 5b pop %ebx

80485a6: 5e pop %esi

80485a7: 5f pop %edi

80485a8: 5d pop %ebp

80485a9: c3 ret

80485aa: 8b 1c 24 mov (%esp),%ebx

80485ad: c3 ret

80485ae: 90 nop

80485af: 90 nop

80485b0: 55 push %ebp

80485b1: 89 e5 mov %esp,%ebp

80485b3: 53 push %ebx

80485b4: 83 ec 04 sub $0x4,%esp

80485b7: a1 f4 96 04 08 mov 0x80496f4,%eax

80485bc: 83 f8 ff cmp $0xffffffff,%eax

80485bf: 74 13 je 80485d4 <tolower@plt+0x24c>

80485c1: bb f4 96 04 08 mov $0x80496f4,%ebx

80485c6: 66 90 xchg %ax,%ax

80485c8: 83 eb 04 sub $0x4,%ebx

80485cb: ff d0 call \*%eax

80485cd: 8b 03 mov (%ebx),%eax

80485cf: 83 f8 ff cmp $0xffffffff,%eax

80485d2: 75 f4 jne 80485c8 <tolower@plt+0x240>

80485d4: 83 c4 04 add $0x4,%esp

80485d7: 5b pop %ebx

80485d8: 5d pop %ebp

80485d9: c3 ret

80485da: 90 nop

80485db: 90 nop

Disassembly of section .fini:

080485dc <.fini>:

80485dc: 55 push %ebp

80485dd: 89 e5 mov %esp,%ebp

80485df: 53 push %ebx

80485e0: 83 ec 04 sub $0x4,%esp

80485e3: e8 00 00 00 00 call 80485e8 <tolower@plt+0x260>

80485e8: 5b pop %ebx

80485e9: 81 c3 ec 11 00 00 add $0x11ec,%ebx

80485ef: e8 dc fd ff ff call 80483d0 <tolower@plt+0x48>

80485f4: 59 pop %ecx

80485f5: 5b pop %ebx

80485f6: c9 leave

80485f7: c3 ret