Assignment 4 - Computer Exploits

Assignment 4: Exploiting and hardening vulnerable C programs

In this assignment we will study buffer overflow attack: its causes, implementation, and the defense against it. Buffer overflow attacks were brought to public attention by Elias Levi (a.k.a. "Aleph0"), the CTO and co-founder of SecurityFocus, in his landmark paper "Smashing The Stack For Fun And Profit.

It was the first public, accessible, step-by-step introduction to buffer overflow vulnerabilities. It is by no means secret information: that paper currently has 1316 citations according to Google Scholar.

PLEASE NOTE that any **deliberate** attempt to carry out an **unauthorized** buffer overflow attack against computer system owned or operated by another physical person or legal entity is a **criminal offence** in most jurisdictions including in Ireland. At the same time, it is essential for you as future IT professionals to understand the mechanisms behind such attacks and to ensure that the software you develop is protected against them. That is why we included this exercise in COMP 30080.

1) Read about buffer overflow attack

Familiarize yourself with the "Smashing The Stack For Fun And Profit" article, which can be found at

https://inst.eecs.berkeley.edu/~cs161/fa08/papers/stack_smashing.pdf

Although the attack principles described by Levi are applicable to most computer architectures, he uses Intel x86 CPU assembly language to demonstrate his ideas.

I have adapted his ideas to MIPS Linux environment and provided you with an example of how a simple buffer overflow vulnerability can be carried out against a vulnerable MIPS program. Please study this example alongside the pre-recorded presentation from November 18th, 2020.

2) Download and install GNU C compiler and related utilities for MIPS little endian architecture

You will need to use GNU C compiler and binary utilities for MIPS32 Little Endian (MIPSEL) CPU to compile the vulnerable C program and to develop your shellcode. Therefore, you will need a virtual machine (or a physical PC) running recent version of Ubuntu Linux. All tasks specified in this document had been verified on the **mips-tools** virtual appliance.

To install GNU C compiler for MIPS32 Little Endian CPU, QEMU userland emulator, and multi-architecture GNU debugger. Start a terminal window and run the following commands:

```
sudo apt-get install qemu-user qemu-system-mips
```

```
sudo apt-get install gcc-mipsel-linux-gnu gdb-multiarch
```

For additional information about C calling convention please refer to Chapters 7 of MIPS Assembly Language Programmer's Guide (https://csmoodle.ucd.ie/moodle/mod/url/view.php?id=1617) .

To use GNU debugger, open two terminal windows and navigate each of them to the sub-directory containing your program

1. Start MIPS simulator in one window using command:

   ```
   qemu-mipsel -g 1235 -L /usr/mipsel-linux-gnu/ yourProgram
   ```

   here -g 1235 option specifies TCP port waiting for GNU debugger connection.

2. Start GNU Debugger in the other window using command:

   ```
   gdb-multiarch yourProgram -ex "target remote :1235" -ex "set arch mips"
   ```

3. Once GDB starts switch to the interactive "Text User Interface" mode by entering command

   ```
   tui enable
   ```

   If the source code and debugging information is available, enable display of program source code and CPU registers by entering commands:

```
layout regs
```

alternatively, if the source code and/or debugging information is not available, display program disassembly and CPU registers by entering commands:

```
layout asm
layout regs
```

4. Step through the program **s**ingle **i**nstruction at a time using command

   ```
   si
   ```

   or **s**ingle line (of the source code) at a time using command

   ```
   s
   ```

   to run (continue) program until completion (or the next break point) use command

   ```
   c
   ```

You can set breakpoints using command 'b' please refer to this and other commands' description in the official GDB user manual (https://sourceware.org/gdb/current/onlinedocs/gdb/) and various online tutorials.

3) Download and experiment with provided examples

Download `a4-files.zip` and `example.zip` files provided alongside this assignment and **unzip** them somewhere in your virtual machine.

Study the example of shellcode (`example-sc.s`) provided in the `example.zip` alongside the vulnerable server program, client software and shellcode compilation scripts. Compile the client and server programs using commands

```
mipsel-linux-gnu-gcc --static -fno-stack-protector -g3 -o server server.c
```

```
mipsel-linux-gnu-gcc --static -fno-stack-protector -g3 -o client client.c
```

Start server program using command

```
qemu-mipsel server 8001
```

or simply

```
./server 8001
```

where 8001 is the TCP port that the server is to listen on.

You can give server commands using client program as follows:

```
echo -n "version" | ./client 127.0.0.1 8001
```

where 127.0.0.1 and 8001 are the IP address and TCP port that the server program is listening on.

Assemble shellcode using provided shell script `assemble`

```
bash assemble example-sc
```

Make sure that you DO NOT specify .s extension at the end of the shellcode assembly file. If all goes well, file `example-sc.bin` will be created. You can review its content by disassembling it using provided shell script `disassemble`

```
bash disassemble example-sc.bin
```

If your setup is correct, you should be able to exploit the vulnerability of the server by feeding the shellcode file to the server via the client software using the following command:

```
cat example-sc.bin | ./client 127.0.0.1 8001
```

where 127.0.0.1 and 8001 are the IP address and TCP port that the server program is listening on. This should cause the server program to report an error message and run Linux 'ls' command, which is the deviant behavior induced by the shellcode.

4) Assignment questions

In this assignment you will need to find solutions for two questions.

Q1. Locate folder `q1/` in the `a4-files.zip` It contains a slightly modified version of the example server in which commands are NULL-terminated strings rather than binary blobs. In order to exploit that program, your shellcode MUST NOT contain any zero bytes.

Your task for Q1 is to modify the shellcode program `q1-sc.s` provided in `q1/` so that once assembled it does not contain any 0x00 bytes and can be used to exploit Q1 server as described in the previous section.

To find a solution for Q1 you will have to use MIPS instructions in innovative ways. Here are some **HINTS** for you to consider:

- Although the default NOP instruction has code 0x00000000, you can use any other instruction that does not affect program behavior in place of NOPs.
- .data and .text section sizes must be a multiple of 16 bytes, otherwise the assembler and linker might automatically pad it with 0x00000000s. Check disassembled code.
- `execve()` system call requires its argument strings `cmd`, `amd_arg1`, `cmd_arg2` and the array `exec_args[]` to be NULL-terminated (i.e. strings `cmd`, `amd_arg1`, `cmd_arg2` must be terminated with 0x00 byte and `exec_args[]` with 0x000000 word). The binary shellcode file cannot contain any 0x00 bytes, so you will need to find a way to add these NULL bytes at runtime.
- The starting address of `exec_args[]` must be a multiply of 4. Assembler may automatically prepend `exec_args[]` with some 0x00 bytes otherwise. Watch out for that.
- Read-up about the optional parameter of MIPS syscall instruction (you can use it like "syscall 0x1"). Find out a way to use it!
- More than likely you will need to debug your shellcode with gdb. Please refer to the pre-recorded presentation on how to do it.
- Extend server.c with a line that prints the length of the received data. If your shellcode binary does not contain any 0x00 bytes, the amount of data received by the server should match the size of the shellcode binary file.

Once your modified shellcode is working, include a description of your solution in the report and upload your final version of `q1-sc.s` as the answer to question 1.

[14 Marks]

Q2. Locate folder `q2/` in the `a4-files.zip` Identify and fix the vulnerability in `websrv1.c` program, describe the vulnerability and your method of fixing it in the report. Save the corrected server program as file `q2.c` and submit it as the answer for this question.

[6 Marks]

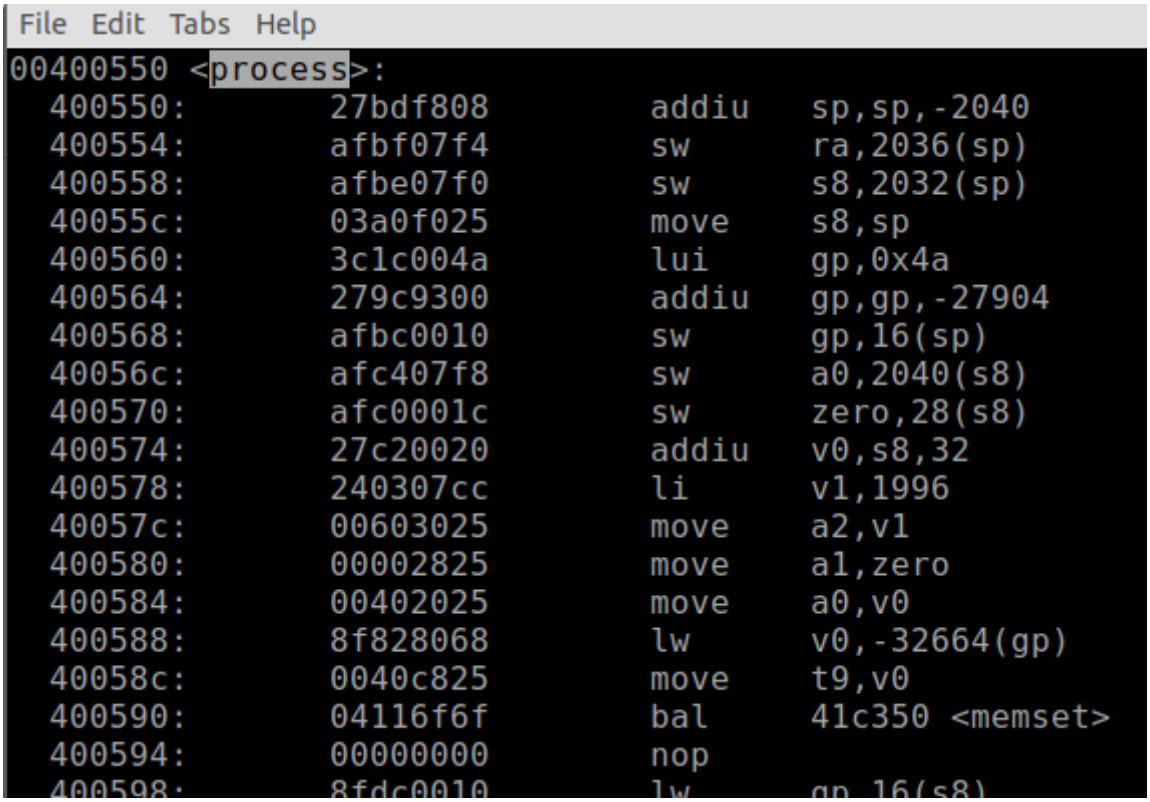
(20 marks in total)


```
mipsel-linux-gnu-gcc –-static -fno-stack-protector -g3 -o server server.c
```

View assembly code

```
mipsel-linux-gnu-objdump -D ./server
// -D dessamble everything
```

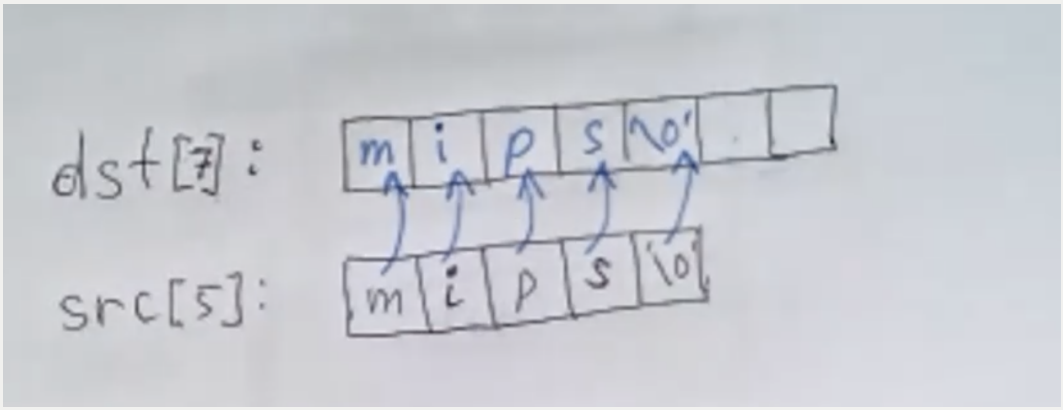
💡 Write to a buffer with a size greater than the size allocated to that buffer ⇒ buffer overflow

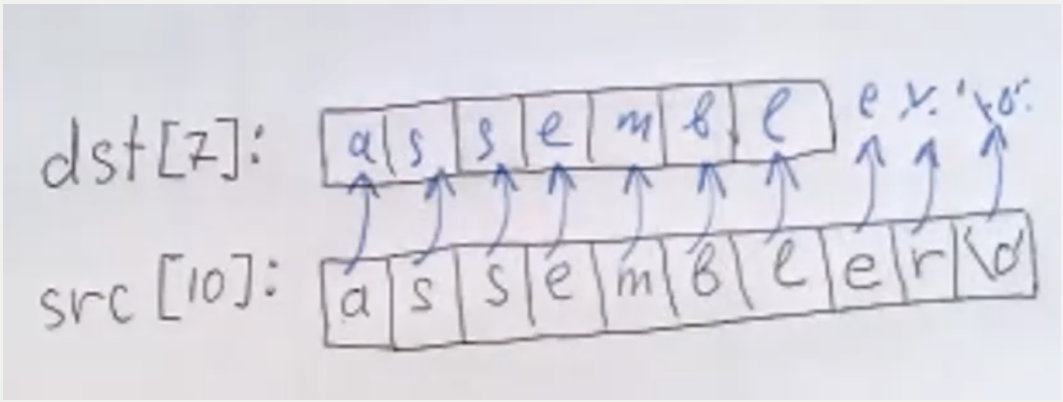


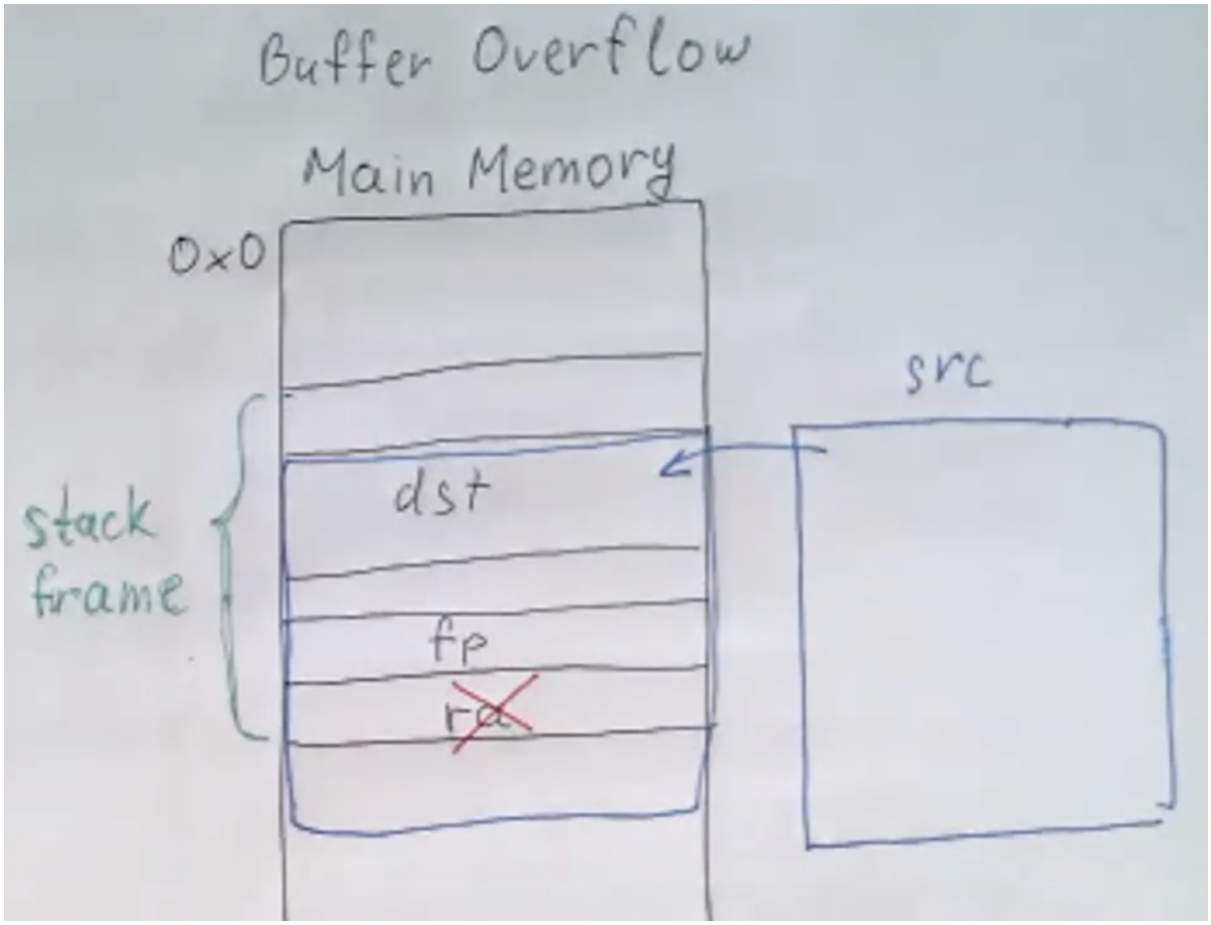
we can see the stack size is 2040

Some C functions are unsafe:

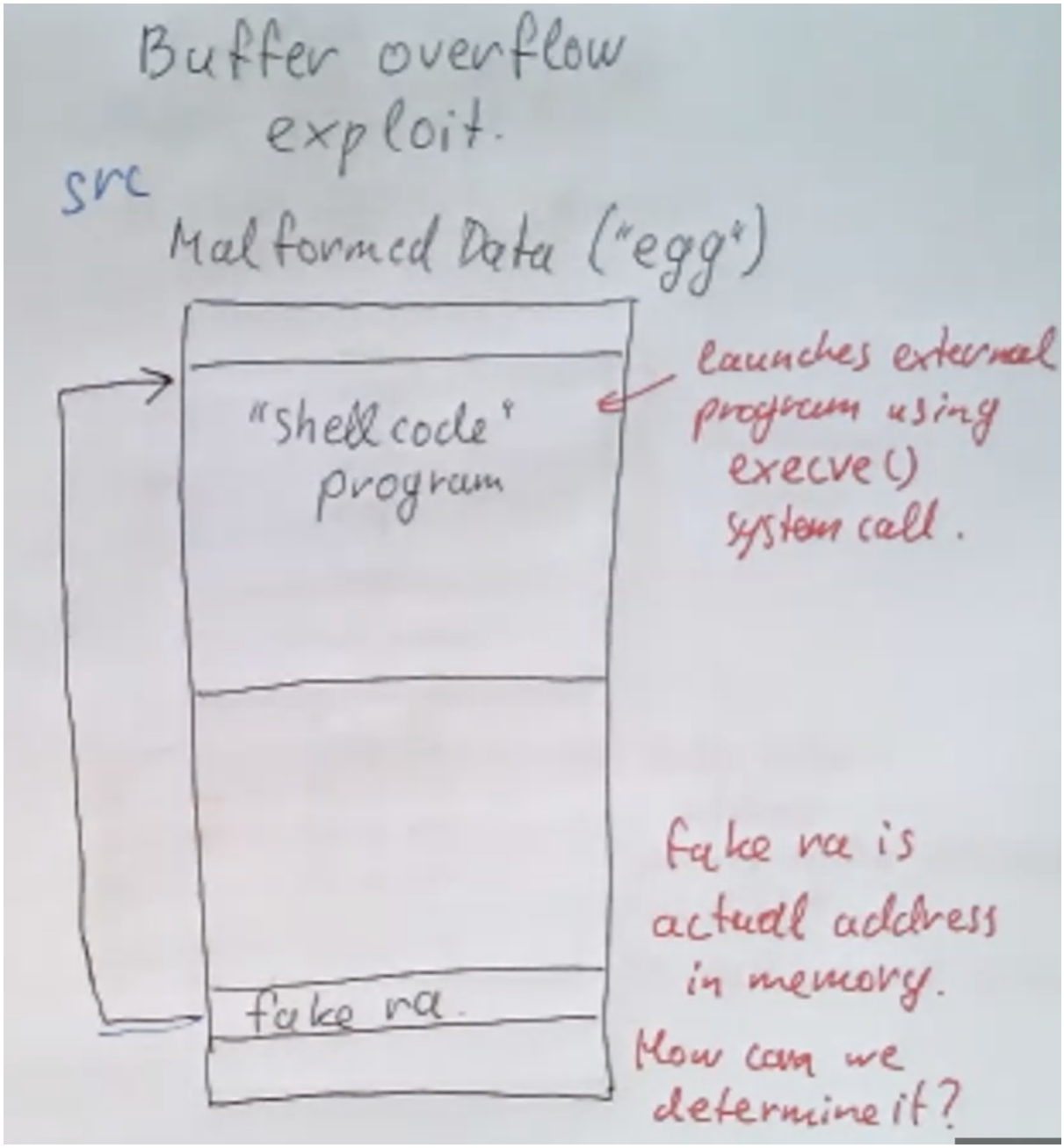
strcpy(dst, src); doesn't check that source array is less than the destination buffer size ⇒ buffer overflow is possible







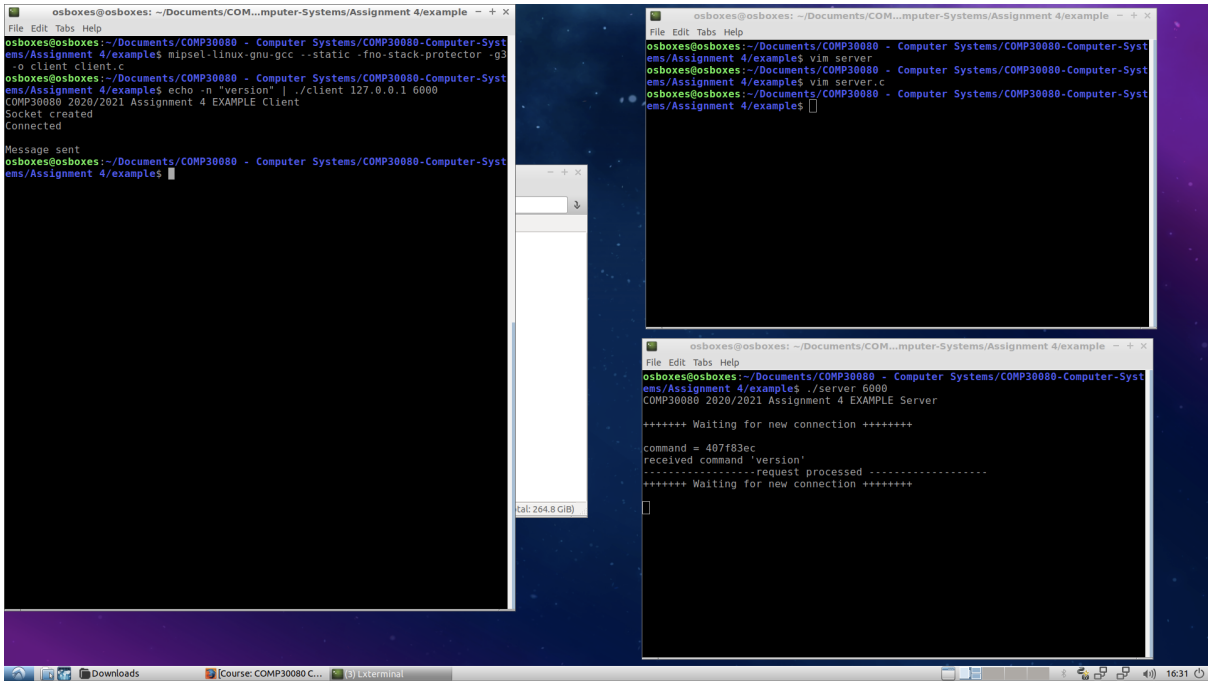
Now we can exploit the return address to run our own code.



Use debugger to step thru the program but there is an easier way,

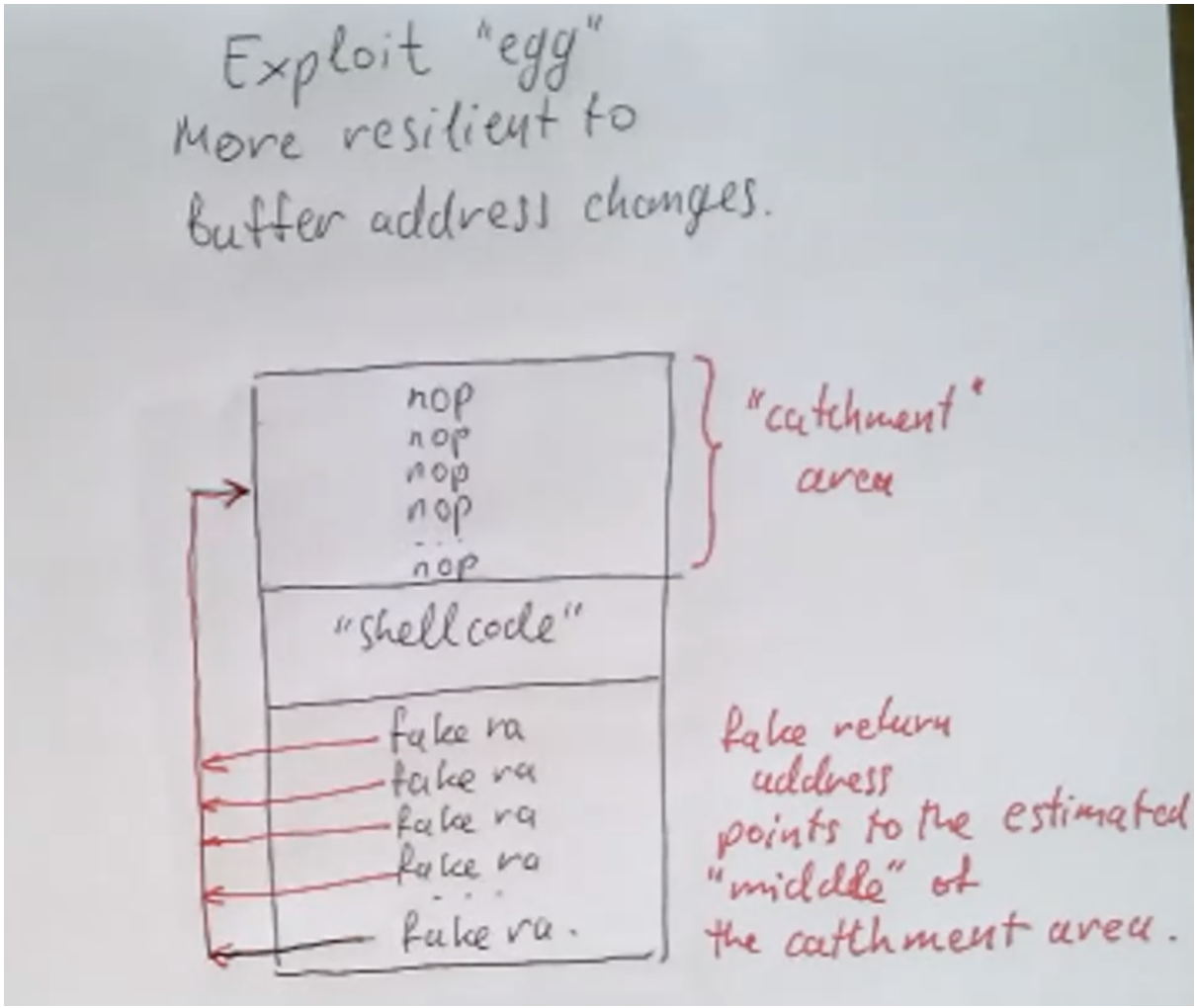
simply inject a printf statement if u have access to the source code

To exploit find the address of the buffer

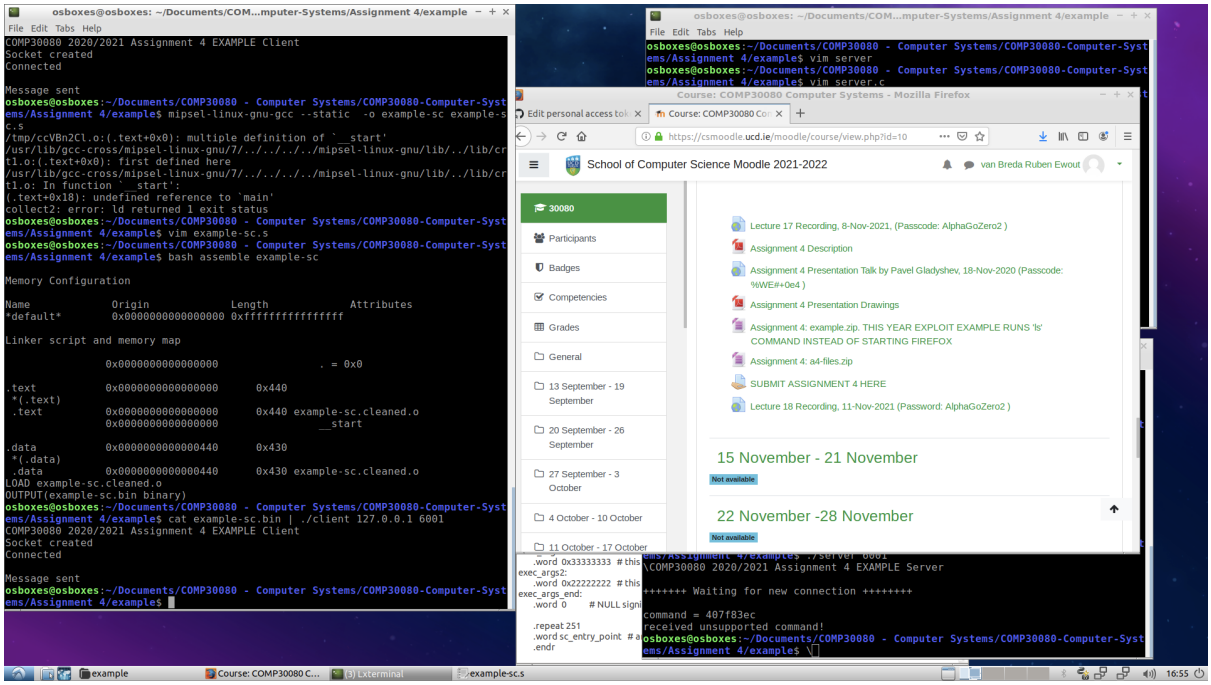


Memory address of exploit buffer can change so we create a catchment area

Catchment area



Exploit to open firefox:



Create the shellcode

A shellcode is a small piece of code used as the payload in the exploitation of a software vulnerability. It is called "shellcode" because it typically starts a command shell from which the attacker can control the compromised machine, but any piece of code that performs a similar task can be called shellcode.

Examing Memory

▼

> Memory (Debugging with GDB)
>
> You can use the command x (for "examine") to examine memory in any of several formats, independently of your program's data types. n, f, and u are all optional parameters that specify how much memory to display and how to format it; addr is an expression giving the address where you
>
> https://sourceware.org/gdb/current/onlinedocs/gdb/Memory.html

10.6 Examining Memory

You can use the command `x` (for “examine”) to examine memory in any of several formats, independently of your program’s data types.

`x/ nfu  addr x  addr x` Use the `x` command to examine memory.

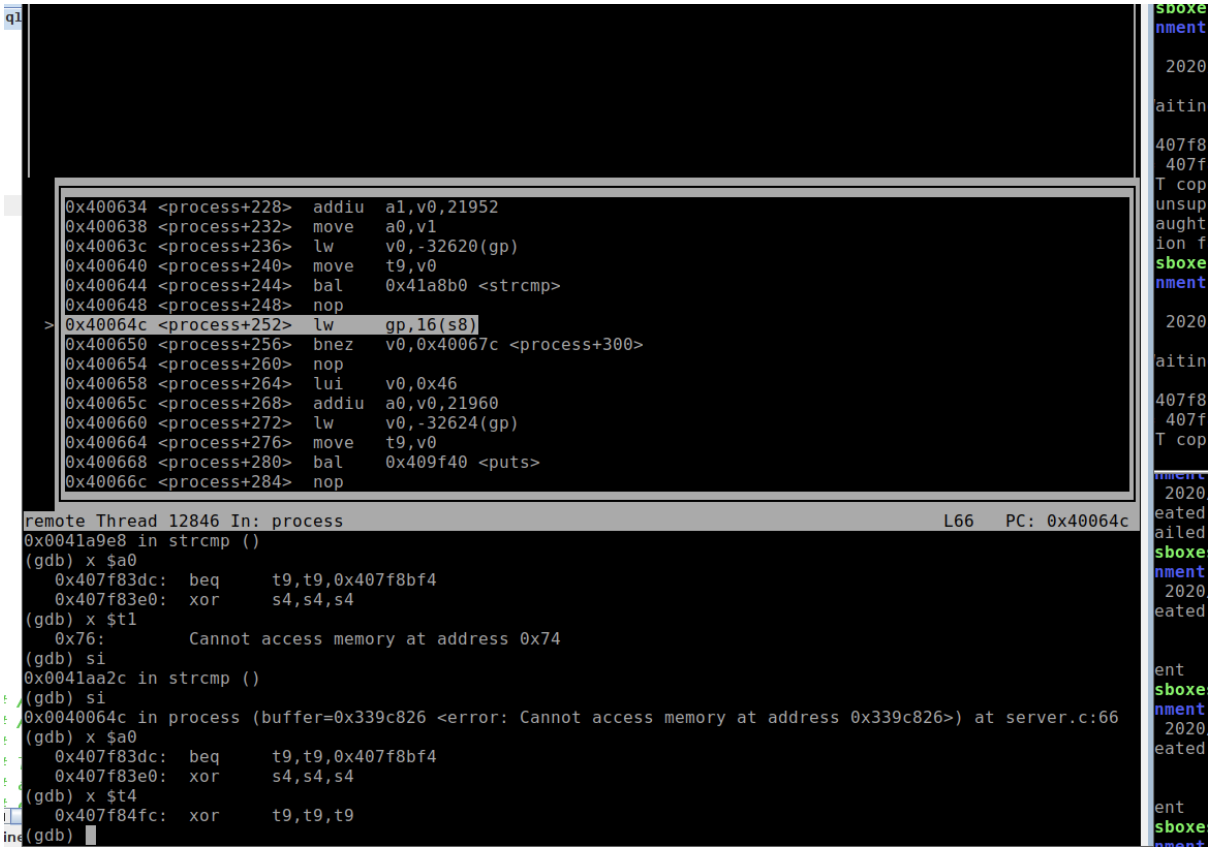
n, f, and u are all optional parameters that specify how much memory to display and how to format it; addr is an expression giving the address where you want to start displaying memory. If you use defaults for nfu, you need not type the slash ‘/’. Several commands set convenient defaults for addr.

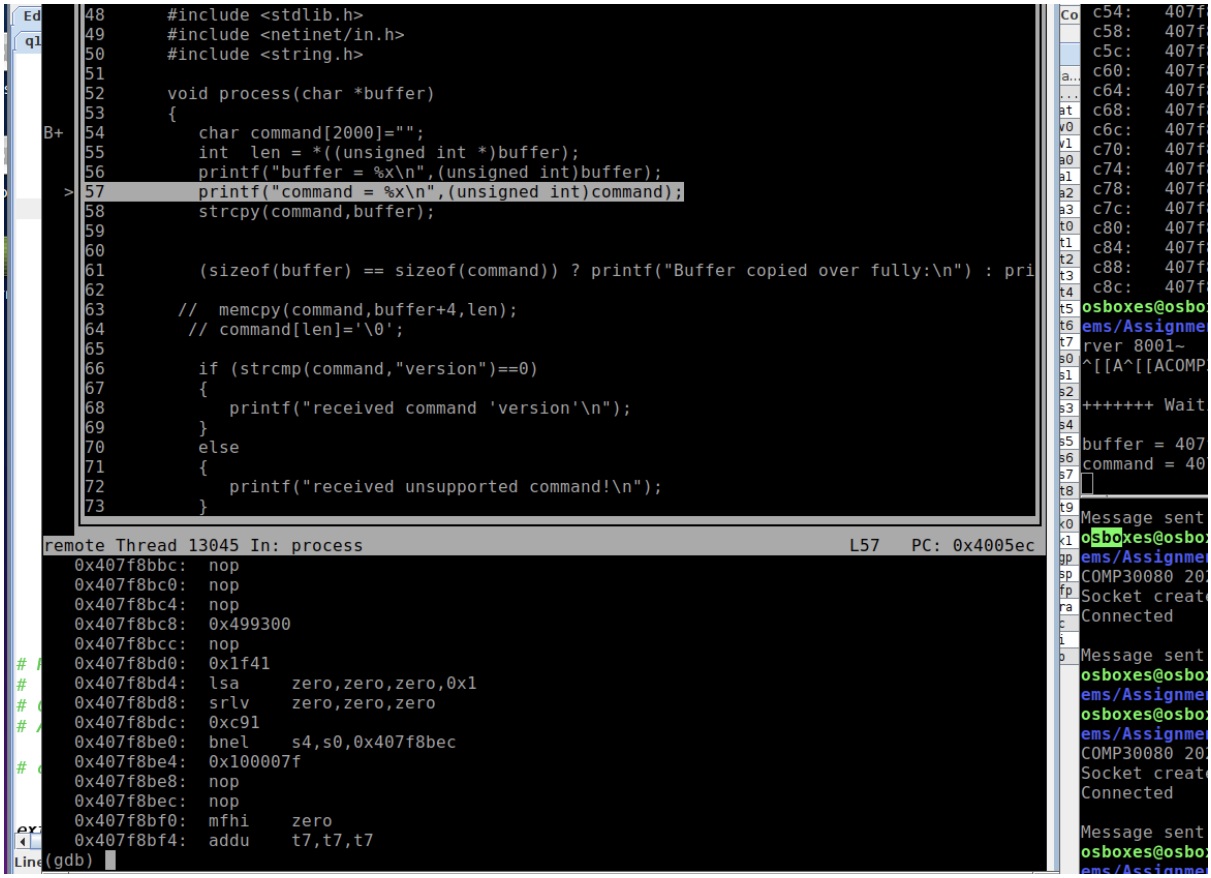
n, the repeat countThe repeat count is a decimal integer; the default is 1. It specifies how much memory (counting by units u) to display. If a negative number is specified, memory is examined backward from addr.f, the display formatThe display format is one of the formats used by `print` (‘x’, ‘d’, ‘u’, ‘o’, ‘t’, ‘a’, ‘c’, ‘f’, ‘s’), ‘i’ (for machine instructions) and ‘m’ (for displaying memory tags). The default is ‘x’ (hexadecimal) initially. The default changes each time you use either `x` or `print`.u, the unit sizeThe unit size is any of`b`Bytes.`h`Halfwords (two bytes).`w`Words (four bytes). This is the initial default.`g`Giant words (eight bytes).

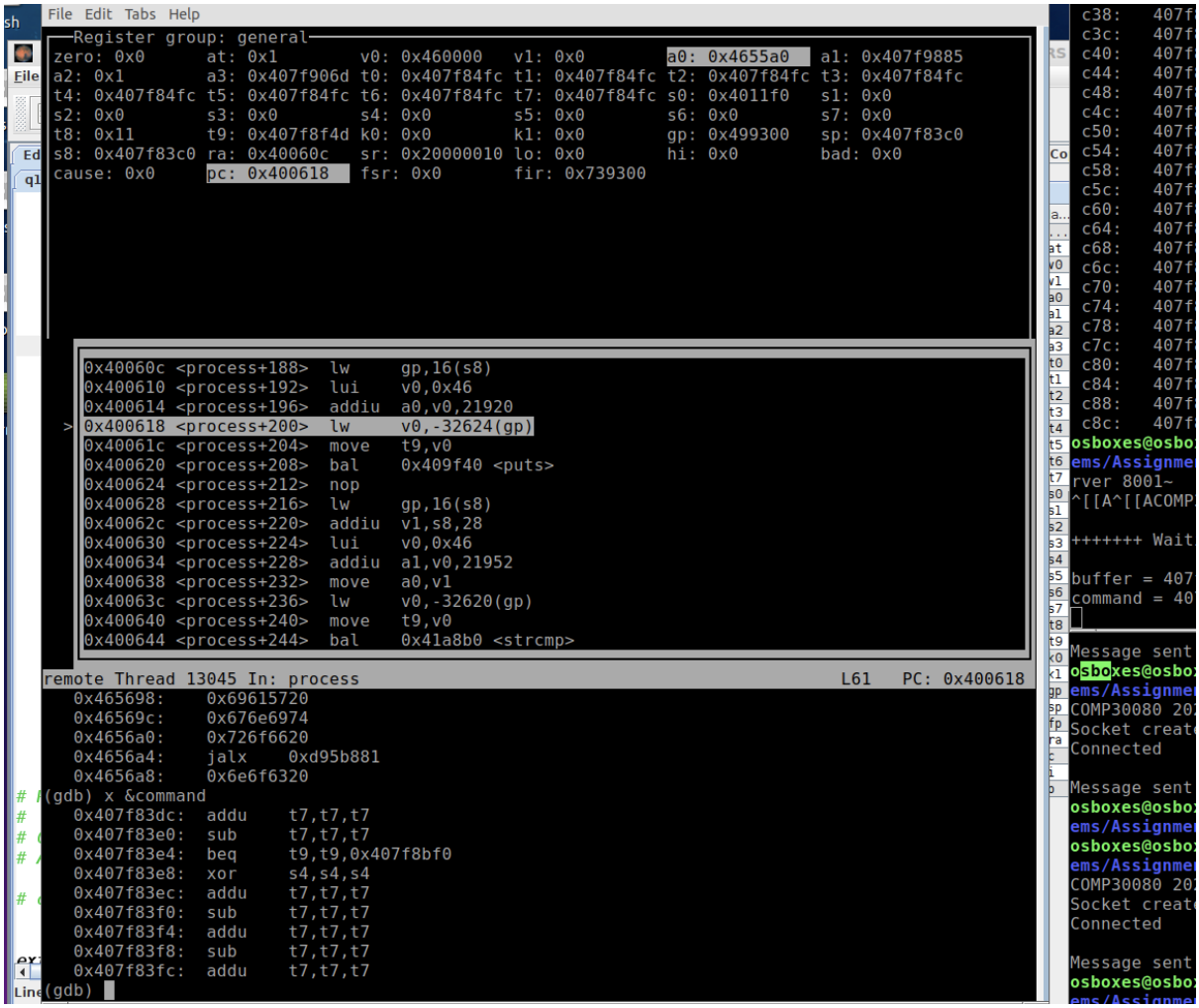
```
x/15c $a0
```

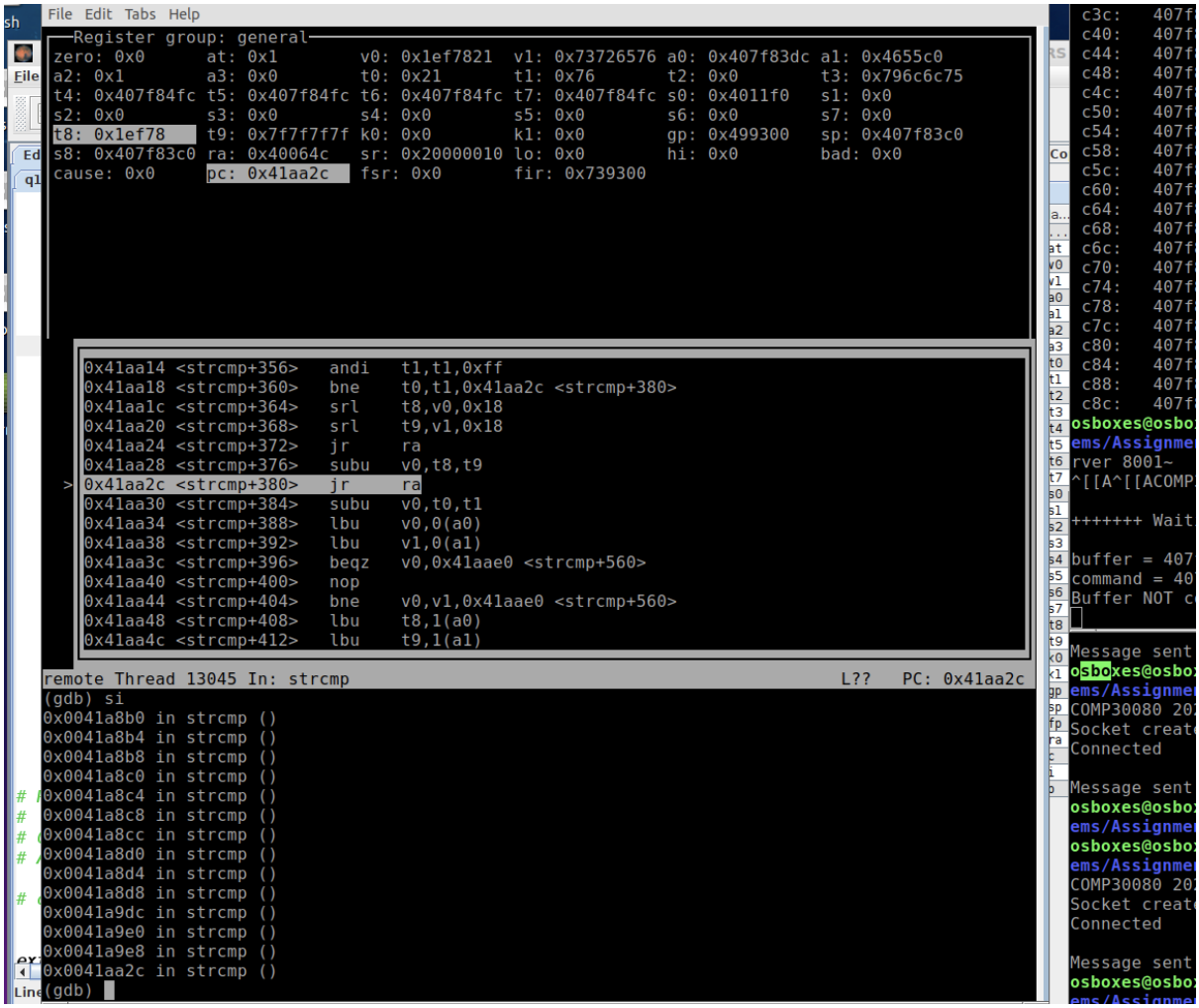
https://sourceware.org/gdb/current/onlinedocs/gdb/Memory.html

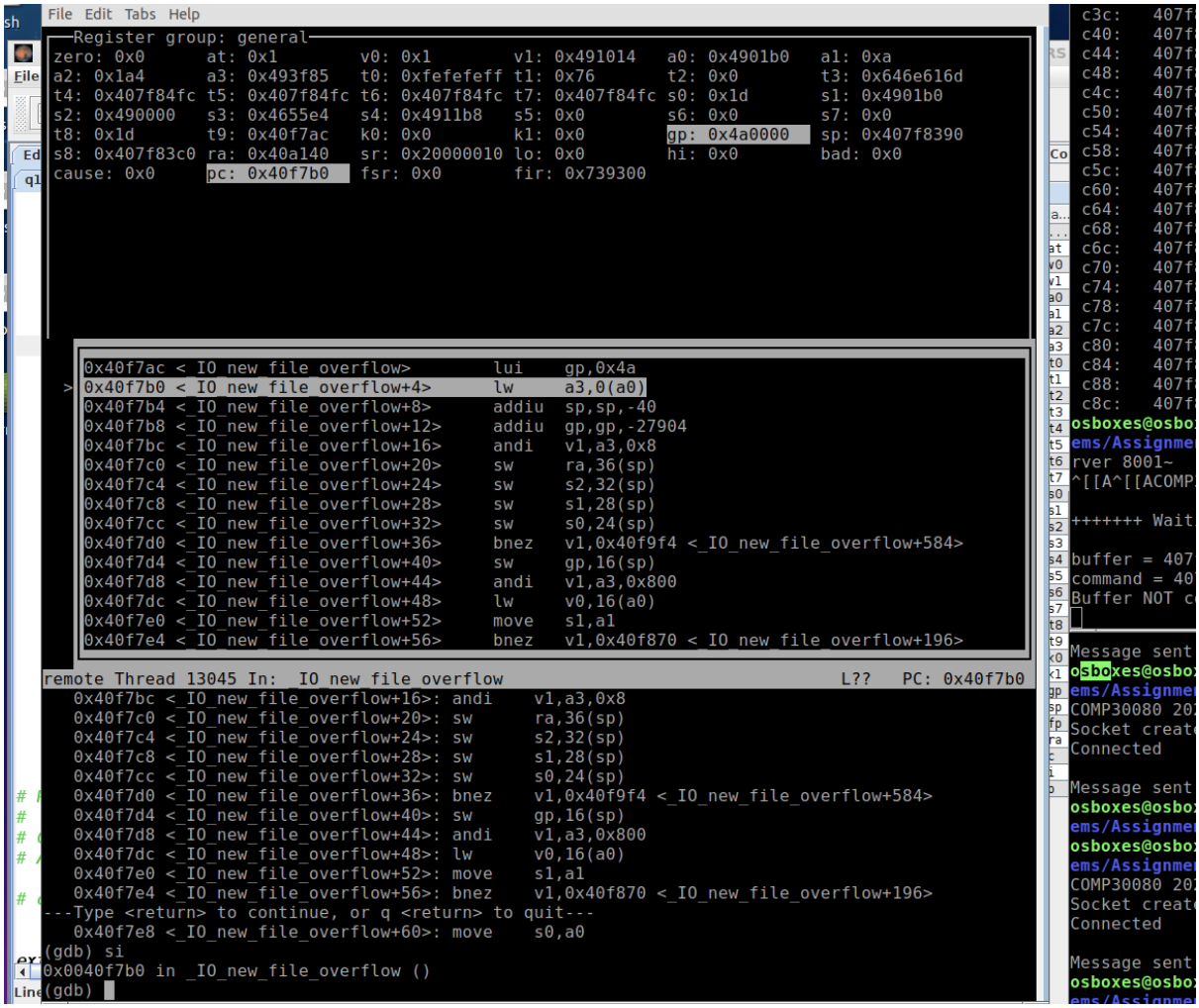
Question 1

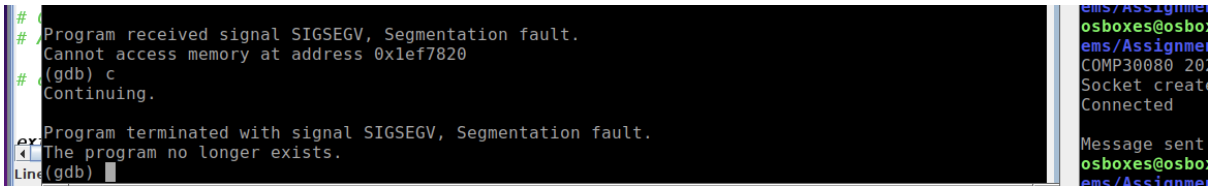

0x400634 <process+228> addiu a1,v0,21952
0x400638 <process+232> move   a0,v1
0x40063c <process+236> lw   v0,-32620(gp)
0x400640 <process+240> move   t9,v0
0x400644 <process+244> bal   0x41a8b0 <strcmp>
0x400648 <process+248> nop
> 0x40064c <process+252> lw   gp,16(s8)
0x400650 <process+256> bnez   v0,0x40067c <process+300>
0x400654 <process+260> nop
0x400658 <process+264> lui   v0,0x46
0x40065c <process+268> addiu a0,v0,21960
0x400660 <process+272> lw   v0,-32624(gp)
0x400664 <process+276> move   t9,v0
0x400668 <process+280> bal   0x409f40 <puts>
0x40066c <process+284> nop
remote Thread 12846 In: process L66 PC: 0x40064c
0x0041a9e8 in strcmp ()
(gdb) x $a0
0x407f83dc: beq t9,t9,0x407f8bf4
0x407f83e0: xor s4,s4,s4
(gdb) x $t1
0x76: Cannot access memory at address 0x74
(gdb) si
0x0041aa2c in strcmp ()
(gdb) si
0x0040064c in process (buffer=0x339c826 <error: Cannot access memory at address 0x339c826>) at server.c:66
(gdb) x $a0
0x407f83dc: beq t9,t9,0x407f8bf4
0x407f83e0: xor s4,s4,s4
(gdb) x $t4
0x407f84fc: xor t9,t9,t9
(gdb)

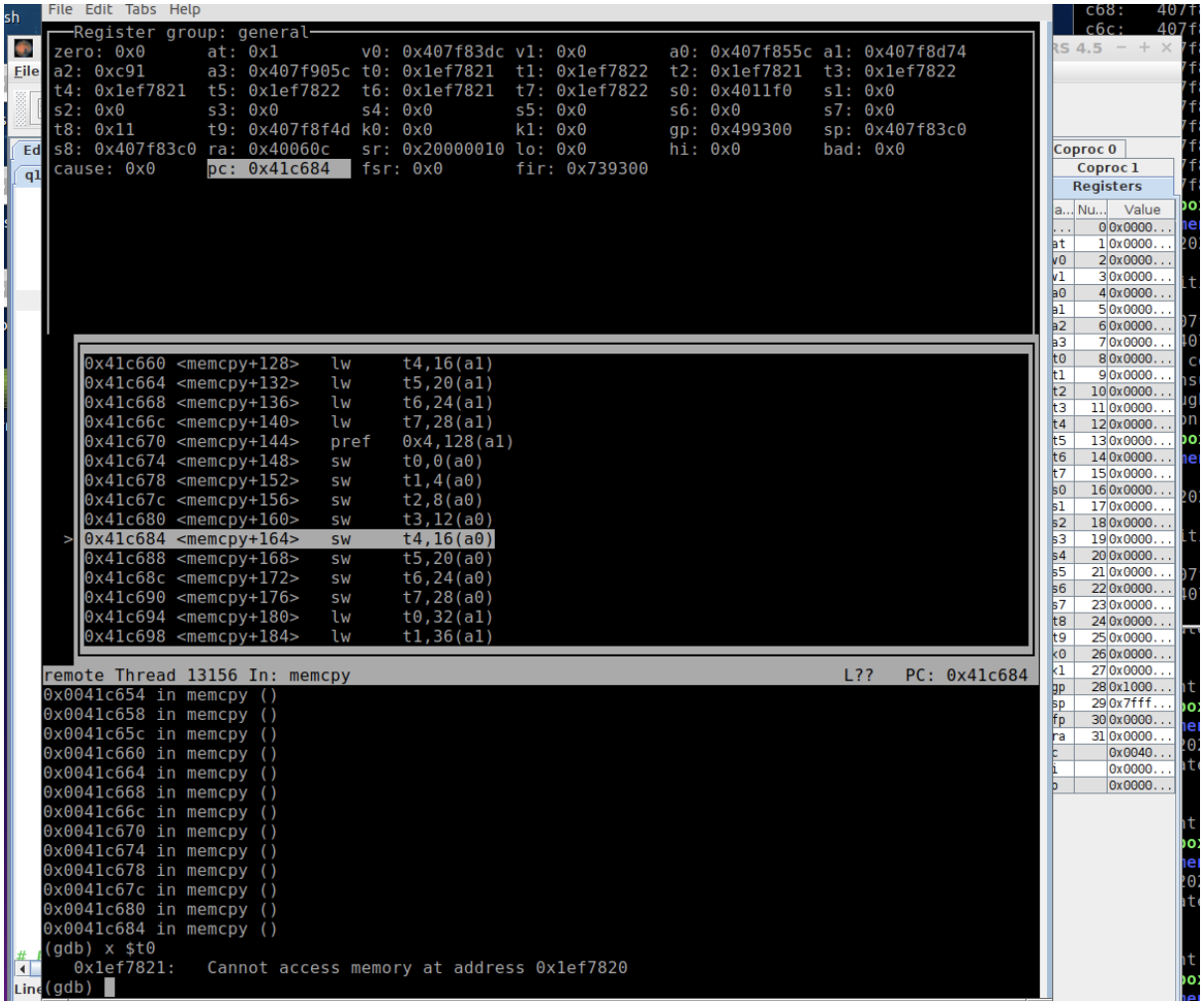

48 #include <stdlib.h>
49 #include <netinet/in.h>
50 #include <string.h>
51
52 void process(char *buffer)
53 {
B+ 54 char command[2000]="";
55 int len = *((unsigned int *)buffer);
56 printf("buffer = %x\n",(unsigned int)buffer);
> 57 printf("command = %x\n",(unsigned int)command);
58 strcpy(command,buffer);
59
60
61 (sizeof(buffer) == sizeof(command)) ? printf("Buffer copied over fully:\n") : pri
62
63 // memcpy(command,buffer+4,len);
64 // command[len]='\0';
65
66 if (strcmp(command,"version")==0)
67 {
68 printf("received command 'version'\n");
69 }
70 else
71 {
72 printf("received unsupported command!\n");
73 }
remote Thread 13045 In: process L57 PC: 0x4005ec
0x407f8bbc: nop
0x407f8bc0: nop
0x407f8bc4: nop
0x407f8bc8: 0x499300
0x407f8bcc: nop
0x407f8bd0: 0x1f41
0x407f8bd4: lsa zero,zero,zero,0x1
0x407f8bd8: srlv zero,zero,zero
0x407f8bdc: 0xc91
0x407f8be0: bnel s4,s0,0x407f8bec
0x407f8be4: 0x100007f
0x407f8be8: nop
0x407f8bec: nop
0x407f8bf0: mfhi zero
0x407f8bf4: addu t7,t7,t7
(gdb)

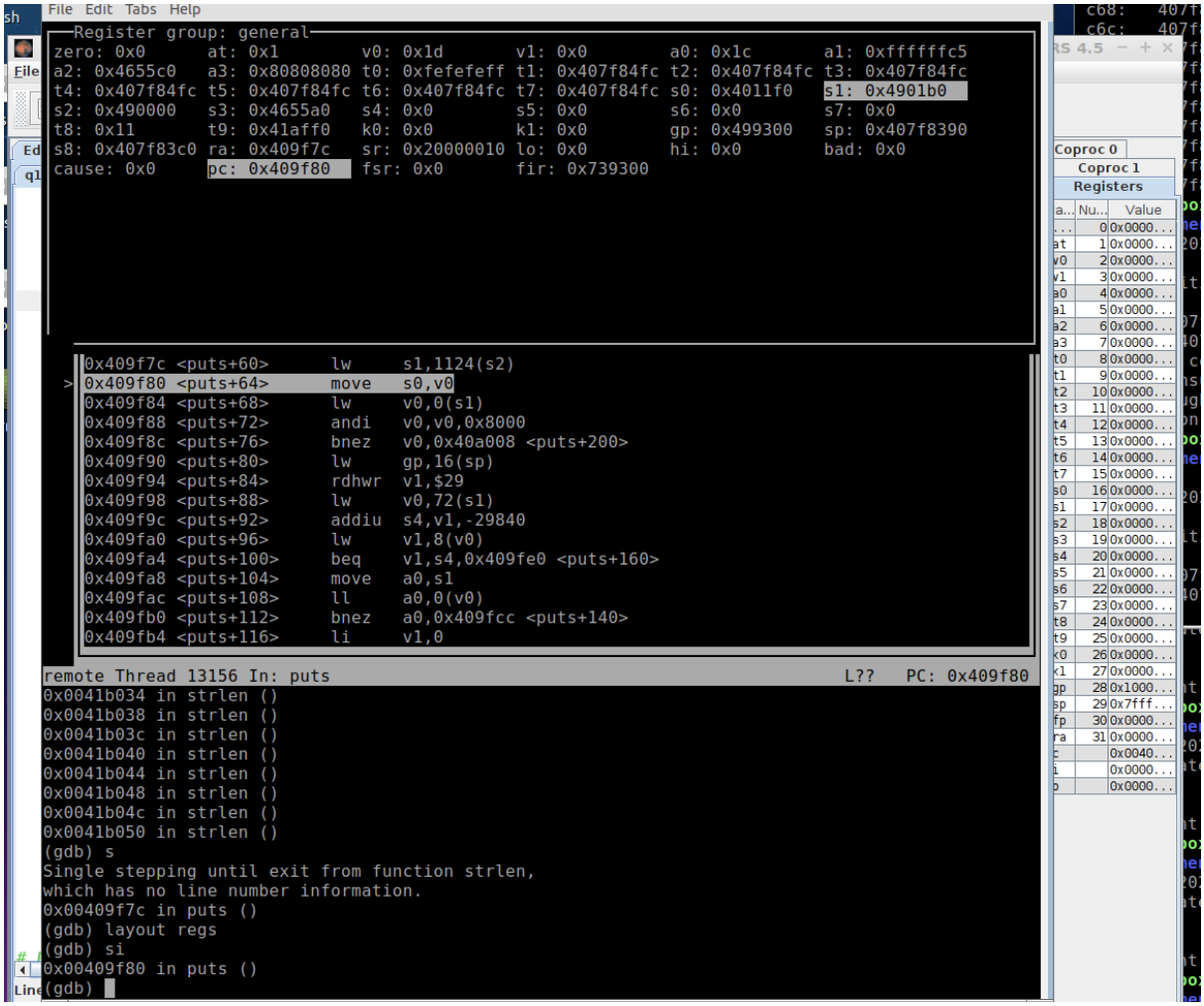

File Edit Tabs Help
Register group: general
zero: 0x0 at: 0x1 v0: 0x460000 v1: 0x0 a0: 0x4655a0 a1: 0x407f9885
a2: 0x1 a3: 0x407f906d t0: 0x407f84fc t1: 0x407f84fc t2: 0x407f84fc t3: 0x407f84fc
t4: 0x407f84fc t5: 0x407f84fc t6: 0x407f84fc t7: 0x407f84fc s0: 0x4011f0 s1: 0x0
s2: 0x0 s3: 0x0 s4: 0x0 s5: 0x0 s6: 0x0 s7: 0x0
t8: 0x11 t9: 0x407f8f4d k0: 0x0 k1: 0x0 gp: 0x499300 sp: 0x407f83c0
s8: 0x407f83c0 ra: 0x40060c sr: 0x20000010 lo: 0x0 hi: 0x0 bad: 0x0
cause: 0x0 pc: 0x400618 fsr: 0x0 fir: 0x739300
0x40060c <process+188> lw gp,16(s8)
0x400610 <process+192> lui v0,0x46
0x400614 <process+196> addiu a0,v0,21920
> 0x400618 <process+200> lw v0,-32624(gp)
0x40061c <process+204> move t9,v0
0x400620 <process+208> bal 0x409f40 <puts>
0x400624 <process+212> nop
0x400628 <process+216> lw gp,16(s8)
0x40062c <process+220> addiu v1,s8,28
0x400630 <process+224> lui v0,0x46
0x400634 <process+228> addiu a1,v0,21952
0x400638 <process+232> move a0,v1
0x40063c <process+236> lw v0,-32620(gp)
0x400640 <process+240> move t9,v0
0x400644 <process+244> bal 0x41a8b0 <strcmp>
remote Thread 13045 In: process L61 PC: 0x400618
0x465698: 0x69615720
0x46569c: 0x676e6974
0x4656a0: 0x726f6620
0x4656a4: jalx 0xd95b881
0x4656a8: 0x6e6f6320
(gdb) x &command
0x407f83dc: addu t7,t7,t7
0x407f83e0: sub t7,t7,t7
0x407f83e4: beq t9,t9,0x407f8bf0
0x407f83e8: xor s4,s4,s4
0x407f83ec: addu t7,t7,t7
0x407f83f0: sub t7,t7,t7
0x407f83f4: addu t7,t7,t7
0x407f83f8: sub t7,t7,t7
0x407f83fc: addu t7,t7,t7
(gdb)
Message sent
Socket creat
Connected
COMP30080 20
osboxes@osbox
ems/Assignme


File Edit Tabs Help
Register group: general
zero: 0x0 at: 0x1 v0: 0x1ef7821 v1: 0x73726576 a0: 0x407f83dc a1: 0x4655c0
a2: 0x1 a3: 0x0 t0: 0x21 t1: 0x76 t2: 0x0 t3: 0x796c6c75
t4: 0x407f84fc t5: 0x407f84fc t6: 0x407f84fc t7: 0x407f84fc s0: 0x4011f0 s1: 0x0
s2: 0x0 s3: 0x0 s4: 0x0 s5: 0x0 s6: 0x0 s7: 0x0
t8: 0x1ef78 t9: 0x7f7f7f7f k0: 0x0 k1: 0x0 gp: 0x499300 sp: 0x407f83c0
s8: 0x407f83c0 ra: 0x40064c sr: 0x20000010 lo: 0x0 hi: 0x0 bad: 0x0
cause: 0x0 pc: 0x41aa2c fsr: 0x0 fir: 0x739300
0x41aa14 <strcmp+356> andi t1,t1,0xff
0x41aa18 <strcmp+360> bne t0,t1,0x41aa2c <strcmp+380>
0x41aa1c <strcmp+364> srl t8,v0,0x18
0x41aa20 <strcmp+368> srl t9,v1,0x18
0x41aa24 <strcmp+372> jr ra
0x41aa28 <strcmp+376> subu v0,t8,t9
> 0x41aa2c <strcmp+380> jr ra
0x41aa30 <strcmp+384> subu v0,t0,t1
0x41aa34 <strcmp+388> lbu v0,0(a0)
0x41aa38 <strcmp+392> lbu v1,0(a1)
0x41aa3c <strcmp+396> beqz v0,0x41aae0 <strcmp+560>
0x41aa40 <strcmp+400> nop
0x41aa44 <strcmp+404> bne v0,v1,0x41aae0 <strcmp+560>
0x41aa48 <strcmp+408> lbu t8,1(a0)
0x41aa4c <strcmp+412> lbu t9,1(a1)
remote Thread 13045 In: strcmp L?? PC: 0x41aa2c
(gdb) si
0x0041a8b0 in strcmp ()
0x0041a8b4 in strcmp ()
0x0041a8b8 in strcmp ()
0x0041a8c0 in strcmp ()
0x0041a8c4 in strcmp ()
0x0041a8c8 in strcmp ()
0x0041a8cc in strcmp ()
0x0041a8d0 in strcmp ()
0x0041a8d4 in strcmp ()
0x0041a8d8 in strcmp ()
0x0041a9dc in strcmp ()
0x0041a9e0 in strcmp ()
0x0041a9e8 in strcmp ()
0x0041aa2c in strcmp ()
(gdb)
c3c: 407f
c40: 407f
c44: 407f
c48: 407f
c4c: 407f
c50: 407f
c54: 407f
c58: 407f
c5c: 407f
c60: 407f
c64: 407f
c68: 407f
c6c: 407f
c70: 407f
c74: 407f
c78: 407f
c7c: 407f
c80: 407f
c84: 407f
c88: 407f
c8c: 407f
osboxes@osbox
ems/Assignme
rver 8001~
^[[A^[[ACOMP
+++++++ Wait
buffer = 407
command = 40
Buffer NOT c
Message sent
osboxes@osbox
ems/Assignme
COMP30080 20
Socket creat
Connected
Message sent
osboxes@osbox
ems/Assignme
osboxes@osbox
ems/Assignme
COMP30080 20
Socket creat
Connected
Message sent
osboxes@osbox


File Edit Tabs Help
Register group: general
zero: 0x0 at: 0x1 v0: 0x1 v1: 0x491014 a0: 0x4901b0 a1: 0xa
a2: 0x1a4 a3: 0x493f85 t0: 0xfefefeff t1: 0x76 t2: 0x0 t3: 0x646e616d
t4: 0x407f84fc t5: 0x407f84fc t6: 0x407f84fc t7: 0x407f84fc s0: 0x1d s1: 0x4901b0
s2: 0x490000 s3: 0x4655e4 s4: 0x4911b8 s5: 0x0 s6: 0x0 s7: 0x0
t8: 0x1d t9: 0x40f7ac k0: 0x0 k1: 0x0 gp: 0x4a0000 sp: 0x407f8390
s8: 0x407f83c0 ra: 0x40a140 sr: 0x20000010 lo: 0x0 hi: 0x0 bad: 0x0
cause: 0x0 pc: 0x40f7b0 fsr: 0x0 fir: 0x739300
0x40f7ac <_IO_new_file_overflow> lui gp,0x4a
> 0x40f7b0 <_IO_new_file_overflow+4> lw a3,0(a0)
0x40f7b4 <_IO_new_file_overflow+8> addiu sp,sp,-40
0x40f7b8 <_IO_new_file_overflow+12> addiu gp,gp,-27904
0x40f7bc <_IO_new_file_overflow+16> andi v1,a3,0x8
0x40f7c0 <_IO_new_file_overflow+20> sw ra,36(sp)
0x40f7c4 <_IO_new_file_overflow+24> sw s2,32(sp)
0x40f7c8 <_IO_new_file_overflow+28> sw s1,28(sp)
0x40f7cc <_IO_new_file_overflow+32> sw s0,24(sp)
0x40f7d0 <_IO_new_file_overflow+36> bnez v1,0x40f9f4 <_IO_new_file_overflow+584>
0x40f7d4 <_IO_new_file_overflow+40> sw gp,16(sp)
0x40f7d8 <_IO_new_file_overflow+44> andi v1,a3,0x800
0x40f7dc <_IO_new_file_overflow+48> lw v0,16(a0)
0x40f7e0 <_IO_new_file_overflow+52> move s1,a1
0x40f7e4 <_IO_new_file_overflow+56> bnez v1,0x40f870 <_IO_new_file_overflow+196>
remote Thread 13045 In: _IO_new_file_overflow L?? PC: 0x40f7b0
0x40f7bc <_IO_new_file_overflow+16>: andi v1,a3,0x8
0x40f7c0 <_IO_new_file_overflow+20>: sw ra,36(sp)
0x40f7c4 <_IO_new_file_overflow+24>: sw s2,32(sp)
0x40f7c8 <_IO_new_file_overflow+28>: sw s1,28(sp)
0x40f7cc <_IO_new_file_overflow+32>: sw s0,24(sp)
0x40f7d0 <_IO_new_file_overflow+36>: bnez v1,0x40f9f4 <_IO_new_file_overflow+584>
0x40f7d4 <_IO_new_file_overflow+40>: sw gp,16(sp)
0x40f7d8 <_IO_new_file_overflow+44>: andi v1,a3,0x800
0x40f7dc <_IO_new_file_overflow+48>: lw v0,16(a0)
0x40f7e0 <_IO_new_file_overflow+52>: move s1,a1
0x40f7e4 <_IO_new_file_overflow+56>: bnez v1,0x40f870 <_IO_new_file_overflow+196>
---Type <return> to continue, or q <return> to quit---
0x40f7e8 <_IO_new_file_overflow+60>: move s0,a0
(gdb) si
0x0040f7b0 in _IO_new_file_overflow ()
(gdb)
c3c: 407f
c40: 407f
c44: 407f
c48: 407f
c4c: 407f
c50: 407f
c54: 407f
c58: 407f
c5c: 407f
c60: 407f
c64: 407f
c68: 407f
c6c: 407f
c70: 407f
c74: 407f
c78: 407f
c7c: 407f
c80: 407f
c84: 407f
c88: 407f
c8c: 407f
osboxes@osbox
ems/Assignme
rver 8001~
^[[A^[[ACOMP
+++++++ Wait
buffer = 407
command = 40
Buffer NOT c
Message sent
osboxes@osbox
ems/Assignme
COMP30080 201
Socket creat
Connected
Message sent
osboxes@osbox
ems/Assignme
osboxes@osbox
ems/Assignme
COMP30080 201
Socket creat
Connected
Message sent
osboxes@osbox


Program received signal SIGSEGV, Segmentation fault.
Cannot access memory at address 0x1ef7820
(gdb) c
Continuing.
Program terminated with signal SIGSEGV, Segmentation fault.
The program no longer exists.
(gdb)
osboxes@osbox
ems/Assignme
COMP30080 201
Socket creat
Connected
Message sent
osboxes@osbox


File Edit Tabs Help
Register group: general
zero: 0x0 at: 0x1 v0: 0x407f83dc v1: 0x0 a0: 0x407f855c a1: 0x407f8d74
a2: 0xc91 a3: 0x407f905c t0: 0x1ef7821 t1: 0x1ef7822 t2: 0x1ef7821 t3: 0x1ef7822
t4: 0x1ef7821 t5: 0x1ef7822 t6: 0x1ef7821 t7: 0x1ef7822 s0: 0x4011f0 s1: 0x0
s2: 0x0 s3: 0x0 s4: 0x0 s5: 0x0 s6: 0x0 s7: 0x0
t8: 0x11 t9: 0x407f8f4d k0: 0x0 k1: 0x0 gp: 0x499300 sp: 0x407f83c0
s8: 0x407f83c0 ra: 0x40060c sr: 0x20000010 lo: 0x0 hi: 0x0 bad: 0x0
cause: 0x0 pc: 0x41c684 fsr: 0x0 fir: 0x739300
0x41c660 <memcpy+128> lw t4,16(a1)
0x41c664 <memcpy+132> lw t5,20(a1)
0x41c668 <memcpy+136> lw t6,24(a1)
0x41c66c <memcpy+140> lw t7,28(a1)
0x41c670 <memcpy+144> pref 0x4,128(a1)
0x41c674 <memcpy+148> sw t0,0(a0)
0x41c678 <memcpy+152> sw t1,4(a0)
0x41c67c <memcpy+156> sw t2,8(a0)
0x41c680 <memcpy+160> sw t3,12(a0)
> 0x41c684 <memcpy+164> sw t4,16(a0)
0x41c688 <memcpy+168> sw t5,20(a0)
0x41c68c <memcpy+172> sw t6,24(a0)
0x41c690 <memcpy+176> sw t7,28(a0)
0x41c694 <memcpy+180> lw t0,32(a1)
0x41c698 <memcpy+184> lw t1,36(a1)
remote Thread 13156 In: memcpy L?? PC: 0x41c684
0x0041c654 in memcpy ()
0x0041c658 in memcpy ()
0x0041c65c in memcpy ()
0x0041c660 in memcpy ()
0x0041c664 in memcpy ()
0x0041c668 in memcpy ()
0x0041c66c in memcpy ()
0x0041c670 in memcpy ()
0x0041c674 in memcpy ()
0x0041c678 in memcpy ()
0x0041c67c in memcpy ()
0x0041c680 in memcpy ()
0x0041c684 in memcpy ()
(gdb) x $t0
0x1ef7821: Cannot access memory at address 0x1ef7820
(gdb)
Coproc 0
Coproc 1
Registers


File Edit Tabs Help
Register group: general
zero: 0x0 at: 0x1 v0: 0x1d v1: 0x0 a0: 0x1c a1: 0xfffffc5
a2: 0x4655c0 a3: 0x80808080 t0: 0xfefefeff t1: 0x407f84fc t2: 0x407f84fc t3: 0x407f84fc
t4: 0x407f84fc t5: 0x407f84fc t6: 0x407f84fc t7: 0x407f84fc s0: 0x4011f0 s1: 0x4901b0
s2: 0x490000 s3: 0x4655a0 s4: 0x0 s5: 0x0 s6: 0x0 s7: 0x0
t8: 0x11 t9: 0x41aff0 k0: 0x0 k1: 0x0 gp: 0x499300 sp: 0x407f8390
s8: 0x407f83c0 ra: 0x409f7c sr: 0x20000010 lo: 0x0 hi: 0x0 bad: 0x0
cause: 0x0 pc: 0x409f80 fsr: 0x0 fir: 0x739300
0x409f7c <puts+60> lw s1,1124(s2)
> 0x409f80 <puts+64> move s0,v0
0x409f84 <puts+68> lw v0,0(s1)
0x409f88 <puts+72> andi v0,v0,0x8000
0x409f8c <puts+76> bnez v0,0x40a008 <puts+200>
0x409f90 <puts+80> lw gp,16(sp)
0x409f94 <puts+84> rdhwr v1,$29
0x409f98 <puts+88> lw v0,72(s1)
0x409f9c <puts+92> addiu s4,v1,-29840
0x409fa0 <puts+96> lw v1,8(v0)
0x409fa4 <puts+100> beq v1,s4,0x409fe0 <puts+160>
0x409fa8 <puts+104> move a0,s1
0x409fac <puts+108> ll a0,0(v0)
0x409fb0 <puts+112> bnez a0,0x409fcc <puts+140>
0x409fb4 <puts+116> li v1,0
remote Thread 13156 In: puts L?? PC: 0x409f80
0x0041b034 in strlen ()
0x0041b038 in strlen ()
0x0041b03c in strlen ()
0x0041b040 in strlen ()
0x0041b044 in strlen ()
0x0041b048 in strlen ()
0x0041b04c in strlen ()
0x0041b050 in strlen ()
(gdb) s
Single stepping until exit from function strlen,
which has no line number information.
0x00409f7c in puts ()
(gdb) layout regs
(gdb) si
0x00409f80 in puts ()
(gdb)
Coproc 0
Coproc 1
Registers