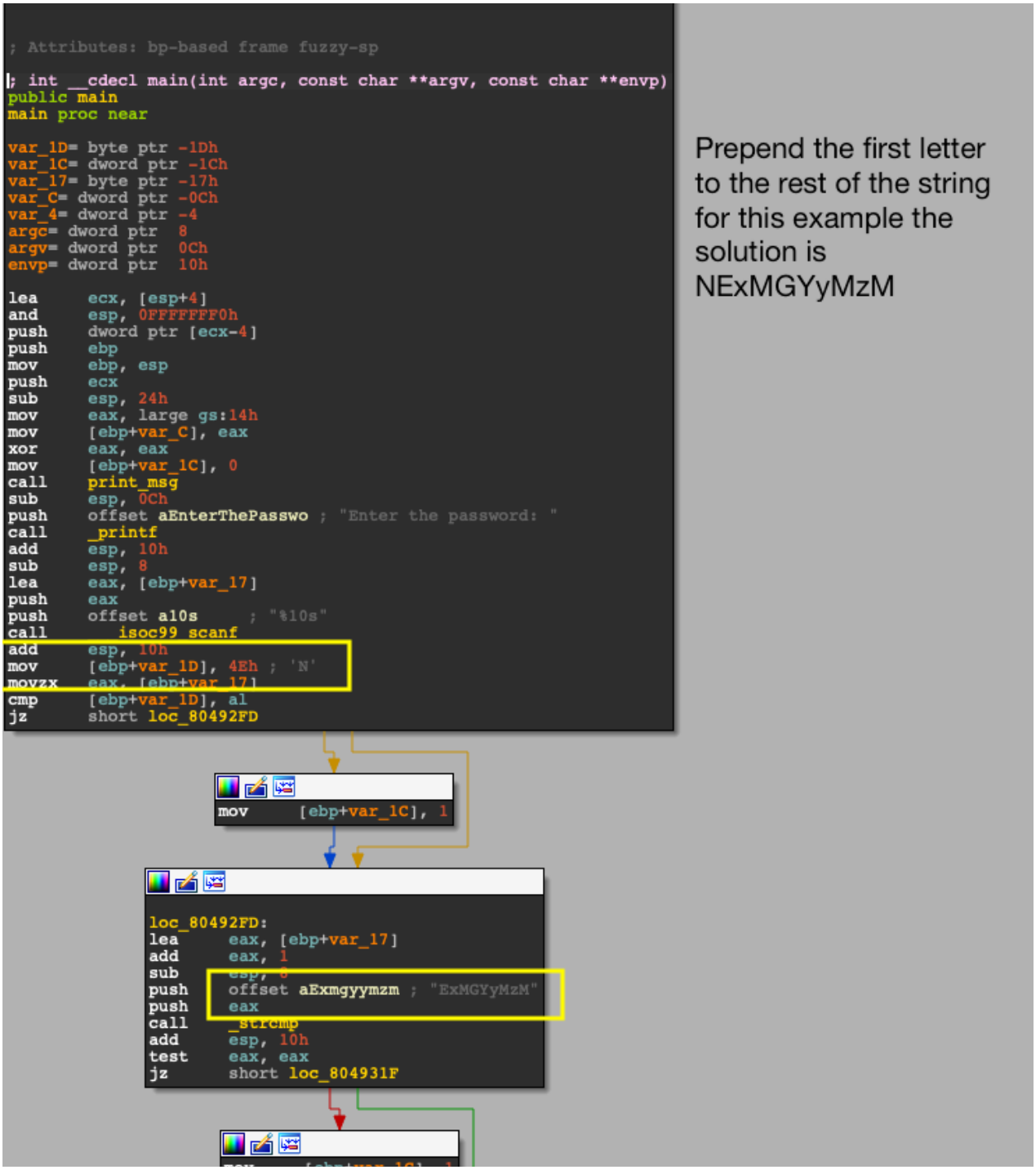
I recommend downloading IDA Pro free version on your local machine or VM as most of the answers are given using that. These screen shots are in order, they cover Chapter 1 -> Ch21 for the Portland State class 'Malware Reverse Engineering' summer 2022 with Wu Chang.

Ch01StatA_Readelf
Ch03DynA_Ltrace
Ch04x86_AsciiInstr
Ch04x86_AsciiStrcmp
Ch06CAsm_Condtionals
Ch06CAsm_LoopMulti
Ch06CAsm_SwitchTable
Ch08Dbg_GdbIntro
Ch08Dbg_GdbParams
Ch08Dbg_GdbPractice
Ch08Dbg_GdbRegs
Ch08Dbg_GdbSetmem
Ch08Dbg_Radare2Intro2
Ch08Dbg_StaticInt
Ch08Dbg_StaticRE
Ch08Dbg_StaticStrcmp
Ch08Dgb_InputFormat
Ch08Dgb_LinkedList
Ch11MalBeh_HijackPLT
Ch11MalBeh_LdPreloadGetUID
Ch11MalBeh_LdPreloadRand
Ch11MalBeh_NetcatShovel
Ch12Convert_ForkFollow
Ch12Covert_ForkPipe
Ch13DataEnc_BaseEnc
Ch13DataEnc_XorEnc
Ch15AntiDis_FakeCallInt
Ch15AntiDis_FakeCond
Ch15AntiDis_FakeMetaConds
Ch15AntiDis_InJmp
Ch15AntiDis_PushRet
Ch16AntiDbg_BypassPtrace
Ch16AntiDbg_GdbCheckTrace
Ch16AntiDbg_Int3Scan
Ch16AntiDbg_SigtrapCheck
Ch16AntiDbg_SigtrapEntangle
Ch16AntiDbg_SigtrapHijack
Ch16AntiDbg_TimeCheck
Ch18PackUnp_UnpackGdb
Ch21x64_ParamsRegs
Ch21x64_ParamsStack

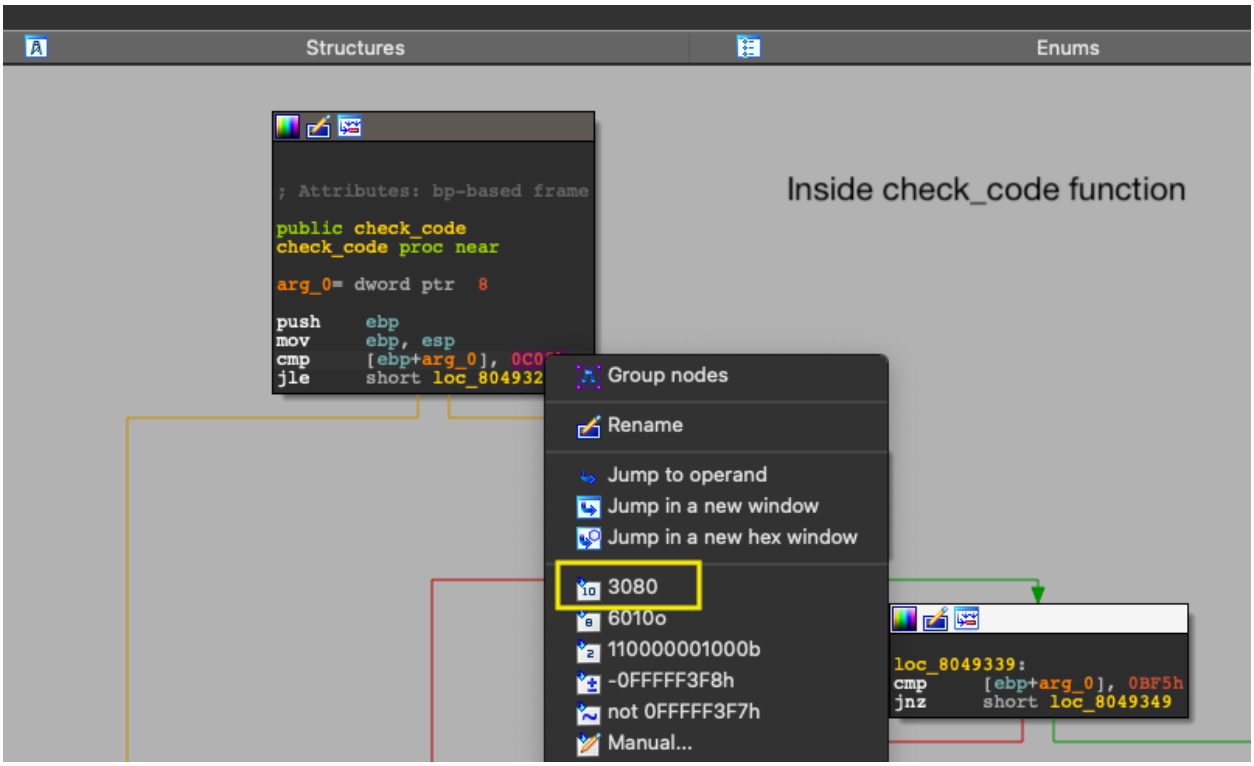

```
jonharr2@jonharr2-VirtualBox:~/metactf/Ch01-08$ ltrace ./Ch03DynA_Ltrace
__libc_start_main(0x804929d, 1, 0xffd0aae4, 0x8049390 <unfinished ...>
strlen("In this level, you will be force"...)     = 311
printf("%s", "In this level, you will be force"...In this level, you will be forced to use dynamic analysis to find the
password.  When programs are dynamically linked, it is possible to monitor
their calls to the libraries they depend upon such as the strcmp() calls
to the Standard C library.  While you can solve this level in many ways try
using "ltrace".

) = 311
printf("Enter the password: ")                    = 20
__isoc99_scanf(0x804a020, 0xffd0aa04, 0xffd0aa38, 0x80492c6Enter the password:  <no return ...>
--- SIGWINCH (Window changed) ---
--- SIGWINCH (Window changed) ---
--- SIGWINCH (Window changed) ---
--- SIGWINCH (Window changed) ---
--- SIGWINCH (Window changed) ---
--- SIGWINCH (Window changed) ---
--- SIGWINCH (Window changed) ---
asdf
<... __isoc99_scanf resumed> )                    = 1
strcmp("asdf", "TYc5MGE1")                        = 1
puts("Try again.Try again.
)                                 = 11
+++ exited (status 0) +++
```

```
; Attributes: bp-based frame fuzzy-sp

; int __cdecl main(int argc, const char **argv, const char **envp)
public main
main proc near

var_24= byte ptr -24h
var_23= byte ptr -23h
var_22= byte ptr -22h
var_21= byte ptr -21h
var_20= byte ptr -20h
var_1F= byte ptr -1Fh
var_1E= byte ptr -1Eh
var_1D= byte ptr -1Dh
var_1C= dword ptr -1Ch
var_15= byte ptr -15h
var_14= byte ptr -14h
var_13= byte ptr -13h
var_12= byte ptr -12h
var_11= byte ptr -11h
var_10= byte ptr -10h
var_F= byte ptr -0Fh
var_E= byte ptr -0Eh
var_C= dword ptr -0Ch
var_4= dword ptr -4
argc= dword ptr  8
argv= dword ptr  0Ch
envp= dword ptr  10h

lea     ecx, [esp+4]
and     esp, 0FFFFFFF0h
push    dword ptr [ecx-4]
push    ebp
mov     ebp, esp
push    ecx
sub     esp, 24h
mov     eax, large gs:14h
mov     [ebp+var_C], eax
xor     eax, eax
mov     [ebp+var_24], 4Eh ; 'N'
mov     [ebp+var_23], 44h ; 'D'
mov     [ebp+var_22], 56h ; 'V'
mov     [ebp+var_21], 6Ah ; 'j'
mov     [ebp+var_20], 5Ah ; 'Z'
mov     [ebp+var_1F], 57h ; 'W'
mov     [ebp+var_1E], 56h ; 'V'
mov     [ebp+var_1D], 6Ah ; 'j'
mov     [ebp+var_1C], 0
call    print_msg
sub     esp, 0Ch
push    offset aEnterThePasswo ; "Enter the password: "
call    _printf
add     esp, 10h
sub     esp, 8
lea     eax, [ebp+var_15]
push    eax
push    offset a8s      ; "%8s"
call    ___isoc99_scanf
add     esp, 10h
movzx   eax, [ebp+var_15]
cmp     [ebp+var_24], al
jz      short loc_8049309
```

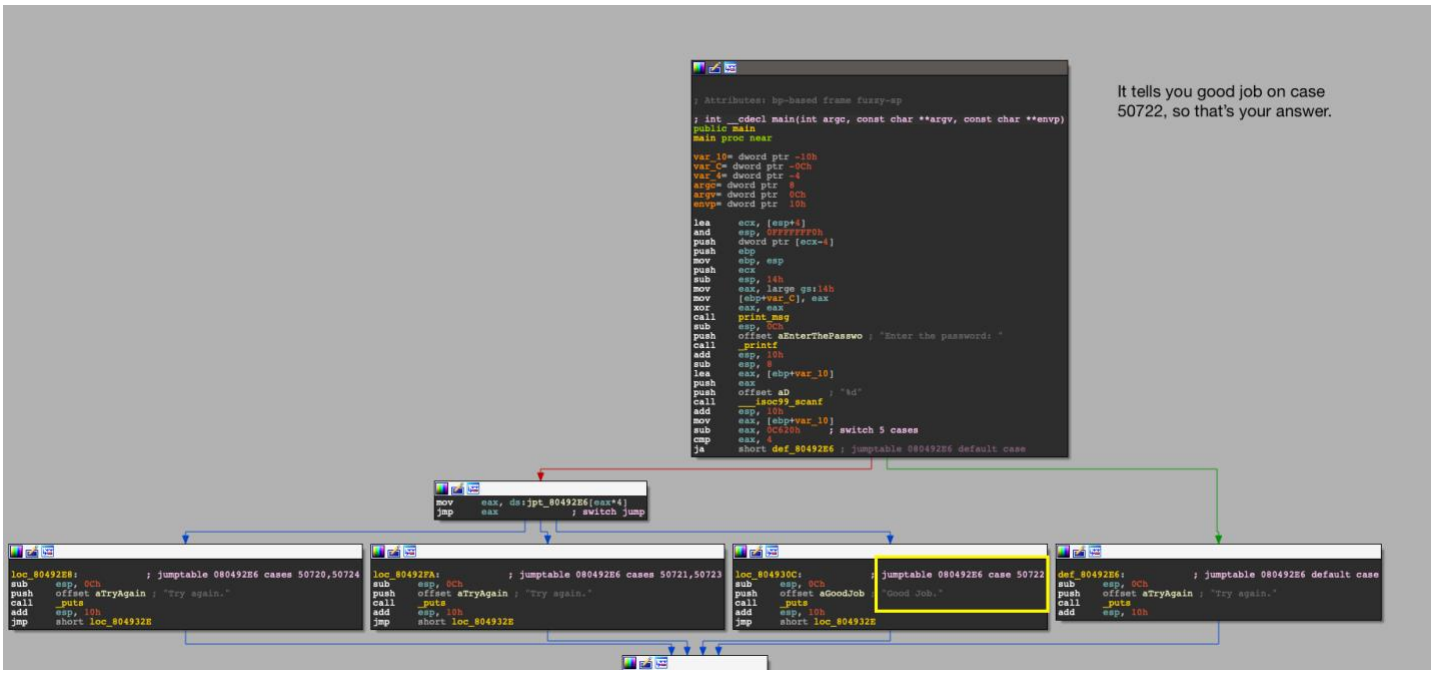
Much easier to look at the code using IDA, we can see the program is moving individual letters into some address space (4EH === 0x4E)



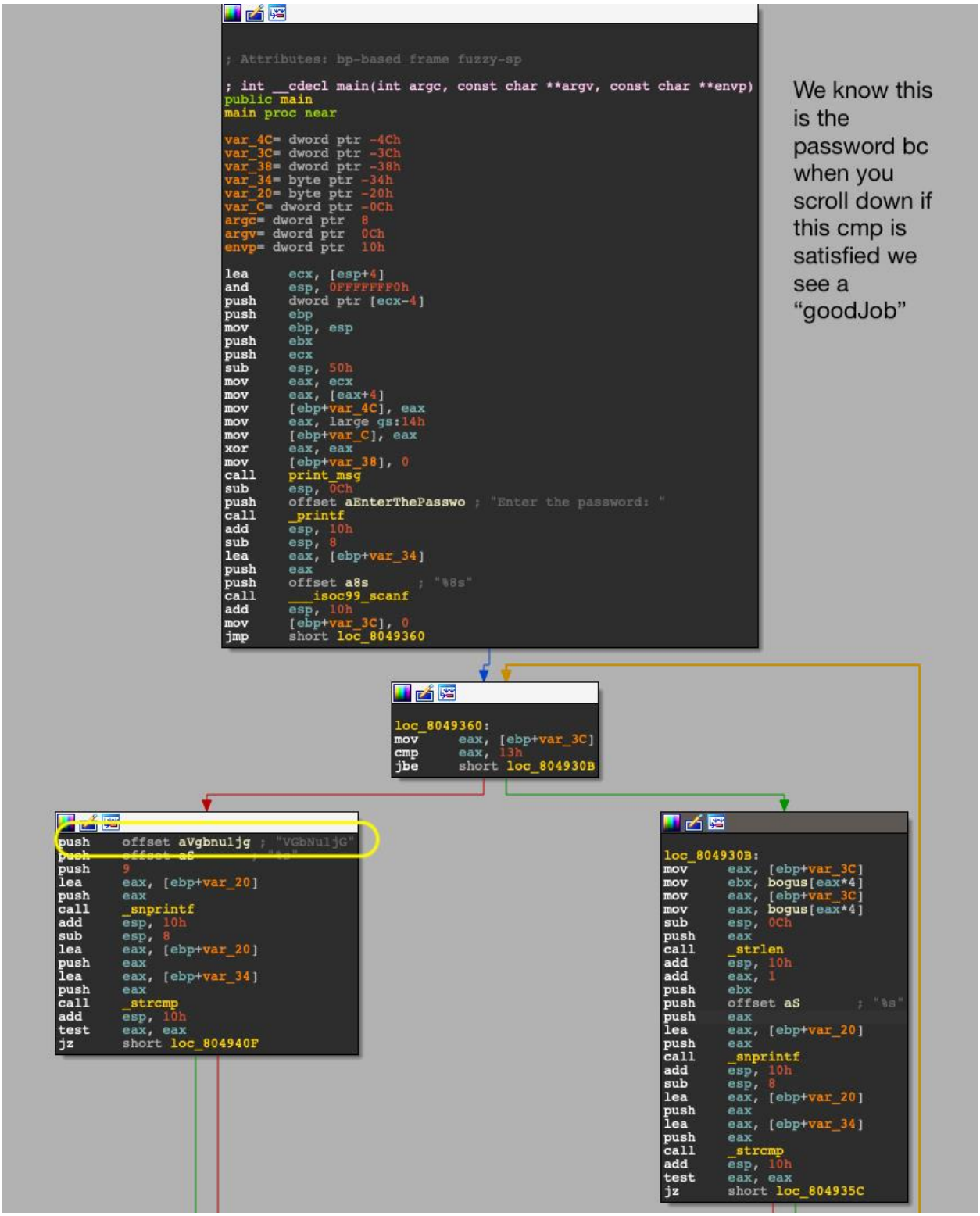
Prepend the first letter to the rest of the string for this example the solution is NExMGYyMzM


Structures
Enums
Inside check_code function
; Attributes: bp-based frame
public check_code
check_code proc near
arg_0= dword ptr 8
push ebp
mov ebp, esp
cmp [ebp+arg_0], 0C0
jle short loc_804932
Group nodes
Rename
Jump to operand
Jump in a new window
Jump in a new hex window
3080
6010o
110000001000b
-0FFFFF3F8h
not 0FFFFF3F7h
Manual...
loc_8049339:
cmp [ebp+arg_0], 0BF5h
jnz short loc_8049349

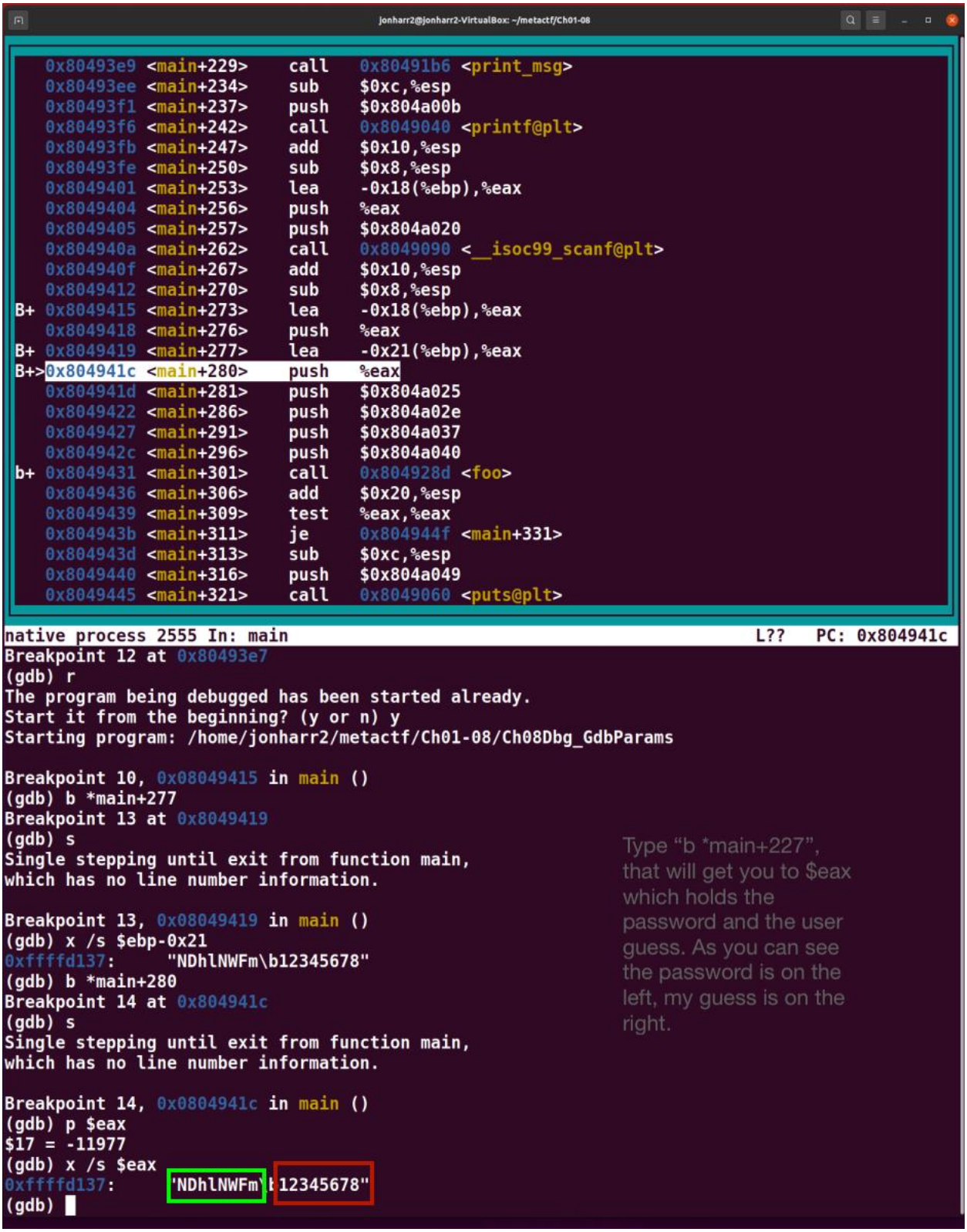

; Attributes: bp-based frame
public loop_multi
loop_multi proc near
var_4= dword ptr -4
arg_0= dword ptr 8
push ebp
mov ebp, esp
sub esp, 10h
mov [ebp+var_4], 0
jmp short loc_8049314
cmp [ebp+var_4], 59h ; 'Y'
jle short loc_8049303
mov [ebp+var_4], 0
jmp short loc_8049334
loc_8049303:
mov eax, [ebp+arg_0]
mov eax, [eax]
lea edx, [eax-1]
mov eax, [ebp+arg_0]
mov [eax], edx
add [ebp+var_4], 1
cmp [ebp+var_4], 0CAh
jle short loc_8049303
mov [ebp+var_4], 0
jmp short loc_8049357
loc_8049323:
mov eax, [ebp+arg_0]
add eax, 4
mov edx, [eax]
sub edx, 1
mov [eax], edx
add [ebp+var_4], 1
loc_8049357:
cmp [ebp+var_4], 26h ; '&'
jle short loc_8049346
loc_8049346:
mov eax, [ebp+arg_0]
add eax, 8
mov edx, [eax]
sub edx, 1
mov [eax], edx
add [ebp+var_4], 1
nop
nop
leave
retn
loop_multi endp
get the hex->decimal of those highlighted values
Add one (cause we start counting at zero)
For this one its 90 203 39



It tells you good job on case 50722, so that's your answer.

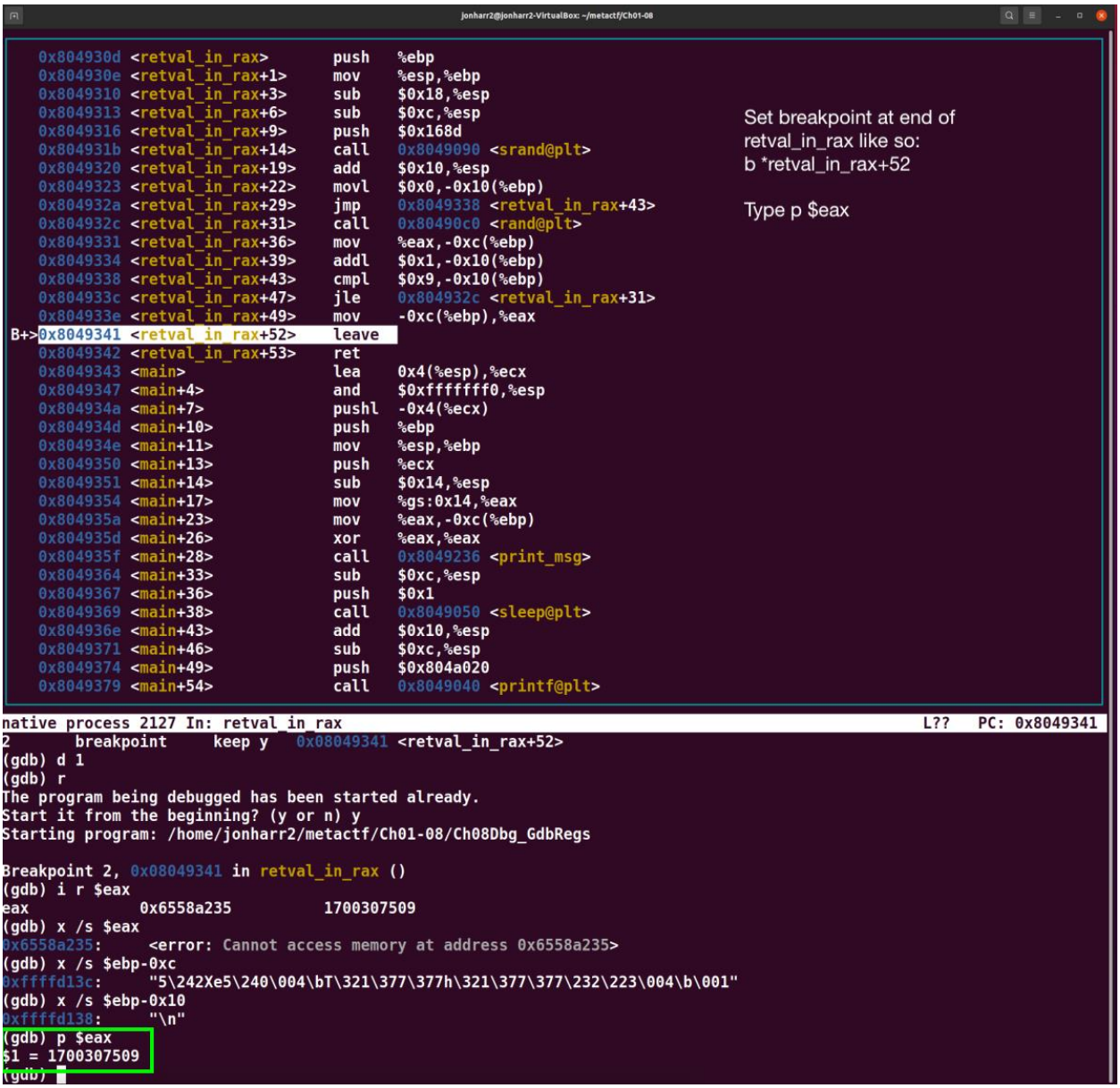


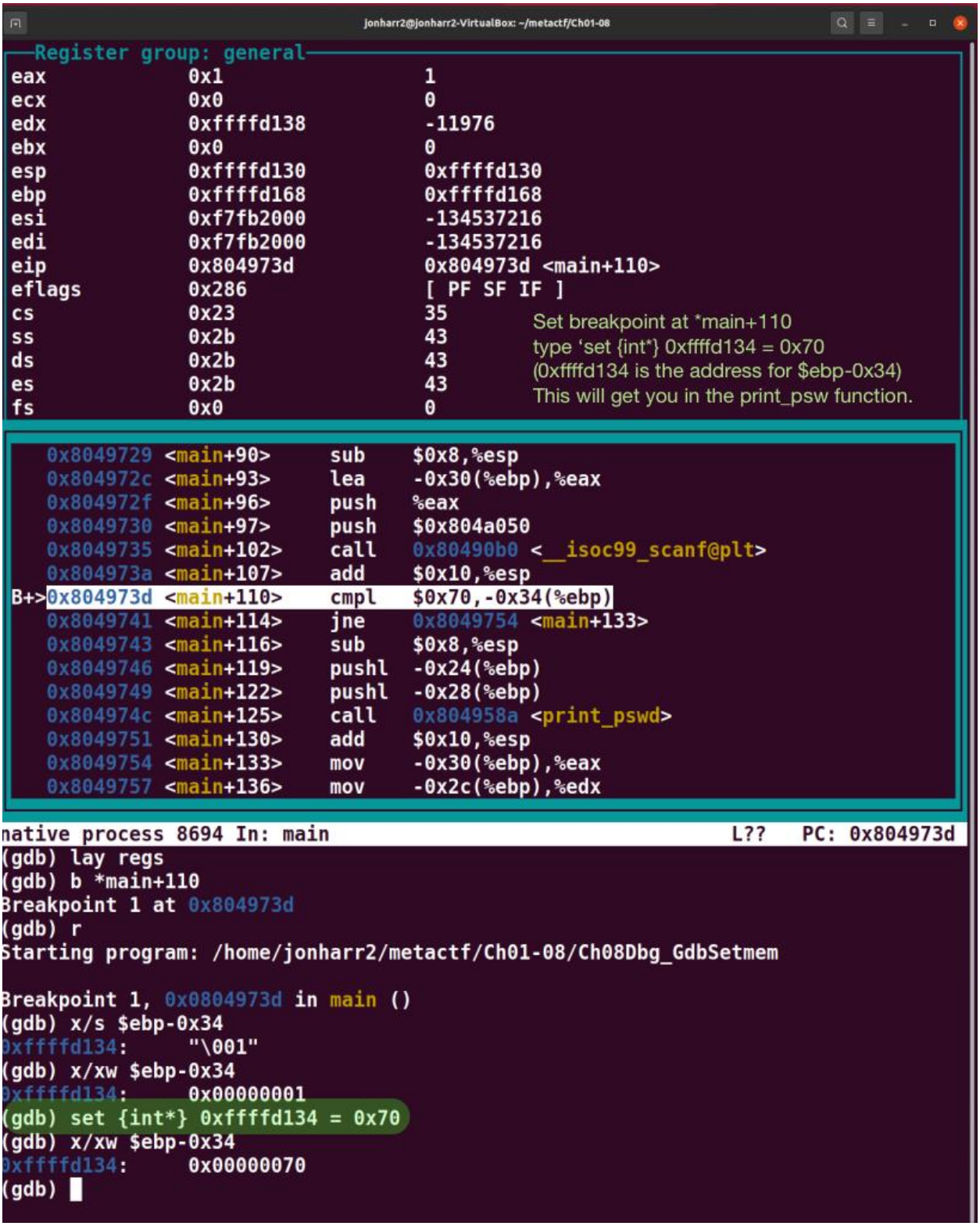
We know this is the password bc when you scroll down if this cmp is satisfied we see a “goodJob”

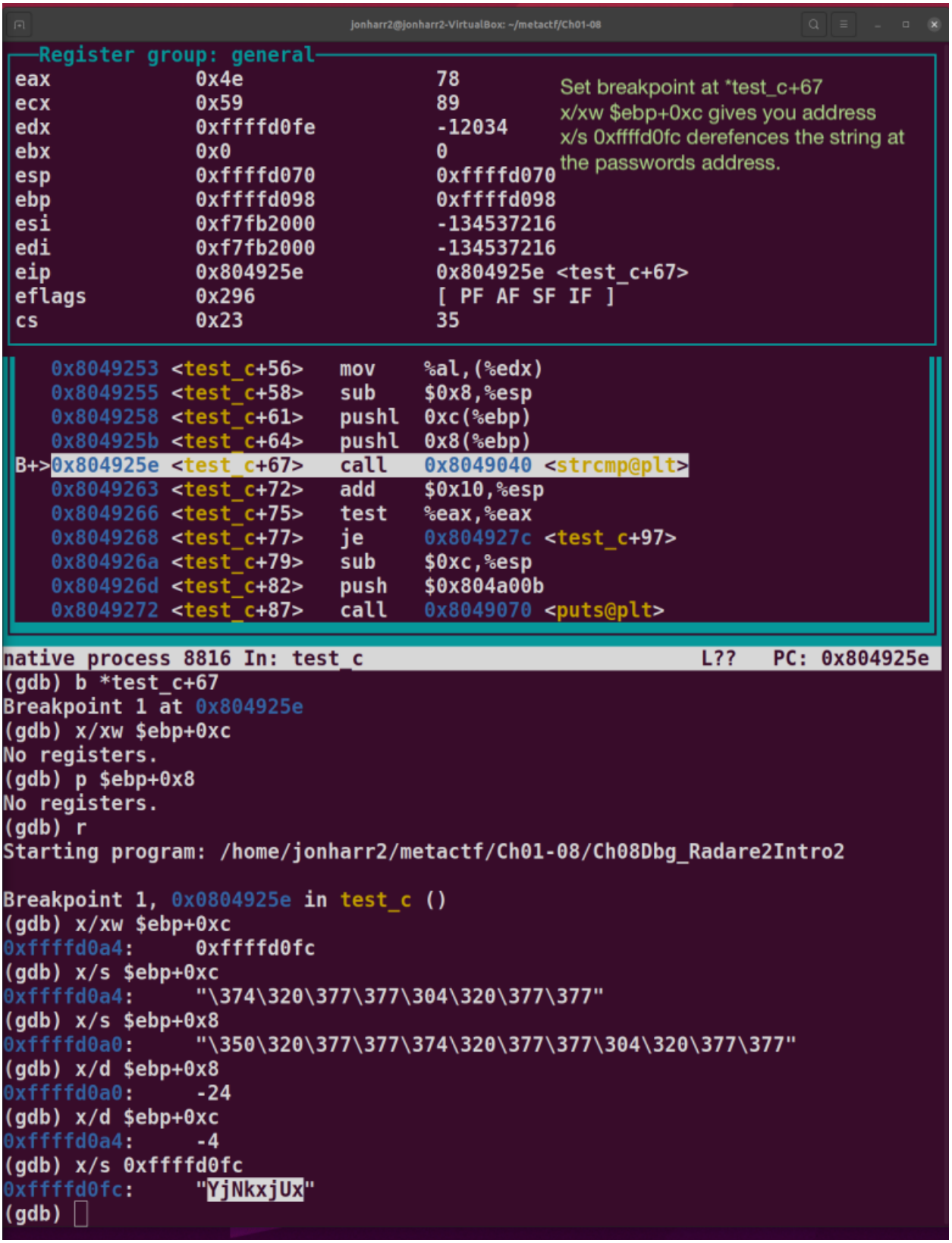

jonharr2@jonharr2-VirtualBox: ~/metactf/Ch01-08
0x80493e9 <main+229> call 0x80491b6 <print_msg>
0x80493ee <main+234> sub $0xc,%esp
0x80493f1 <main+237> push $0x804a00b
0x80493f6 <main+242> call 0x8049040 <printf@plt>
0x80493fb <main+247> add $0x10,%esp
0x80493fe <main+250> sub $0x8,%esp
0x8049401 <main+253> lea -0x18(%ebp),%eax
0x8049404 <main+256> push %eax
0x8049405 <main+257> push $0x804a020
0x804940a <main+262> call 0x8049090 <__isoc99_scanf@plt>
0x804940f <main+267> add $0x10,%esp
0x8049412 <main+270> sub $0x8,%esp
B+ 0x8049415 <main+273> lea -0x18(%ebp),%eax
0x8049418 <main+276> push %eax
B+ 0x8049419 <main+277> lea -0x21(%ebp),%eax
B+> 0x804941c <main+280> push %eax
0x804941d <main+281> push $0x804a025
0x8049422 <main+286> push $0x804a02e
0x8049427 <main+291> push $0x804a037
0x804942c <main+296> push $0x804a040
b+ 0x8049431 <main+301> call 0x804928d <foo>
0x8049436 <main+306> add $0x20,%esp
0x8049439 <main+309> test %eax,%eax
0x804943b <main+311> je 0x804944f <main+331>
0x804943d <main+313> sub $0xc,%esp
0x8049440 <main+316> push $0x804a049
0x8049445 <main+321> call 0x8049060 <puts@plt>
native process 2555 In: main L?? PC: 0x804941c
Breakpoint 12 at 0x80493e7
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/jonharr2/metactf/Ch01-08/Ch08Dbg_GdbParams
Breakpoint 10, 0x08049415 in main ()
(gdb) b *main+277
Breakpoint 13 at 0x8049419
(gdb) s
Single stepping until exit from function main,
which has no line number information.
Breakpoint 13, 0x08049419 in main ()
(gdb) x /s $ebp-0x21
0xffffd137: "NDhlNWFm\b12345678"
(gdb) b *main+280
Breakpoint 14 at 0x804941c
(gdb) s
Single stepping until exit from function main,
which has no line number information.
Breakpoint 14, 0x0804941c in main ()
(gdb) p $eax
$17 = -11977
(gdb) x /s $eax
0xffffd137: "NDhlNWFm\b12345678"
(gdb)
Type “b *main+227”, that will get you to $eax which holds the password and the user guess. As you can see the password is on the left, my guess is on the right.

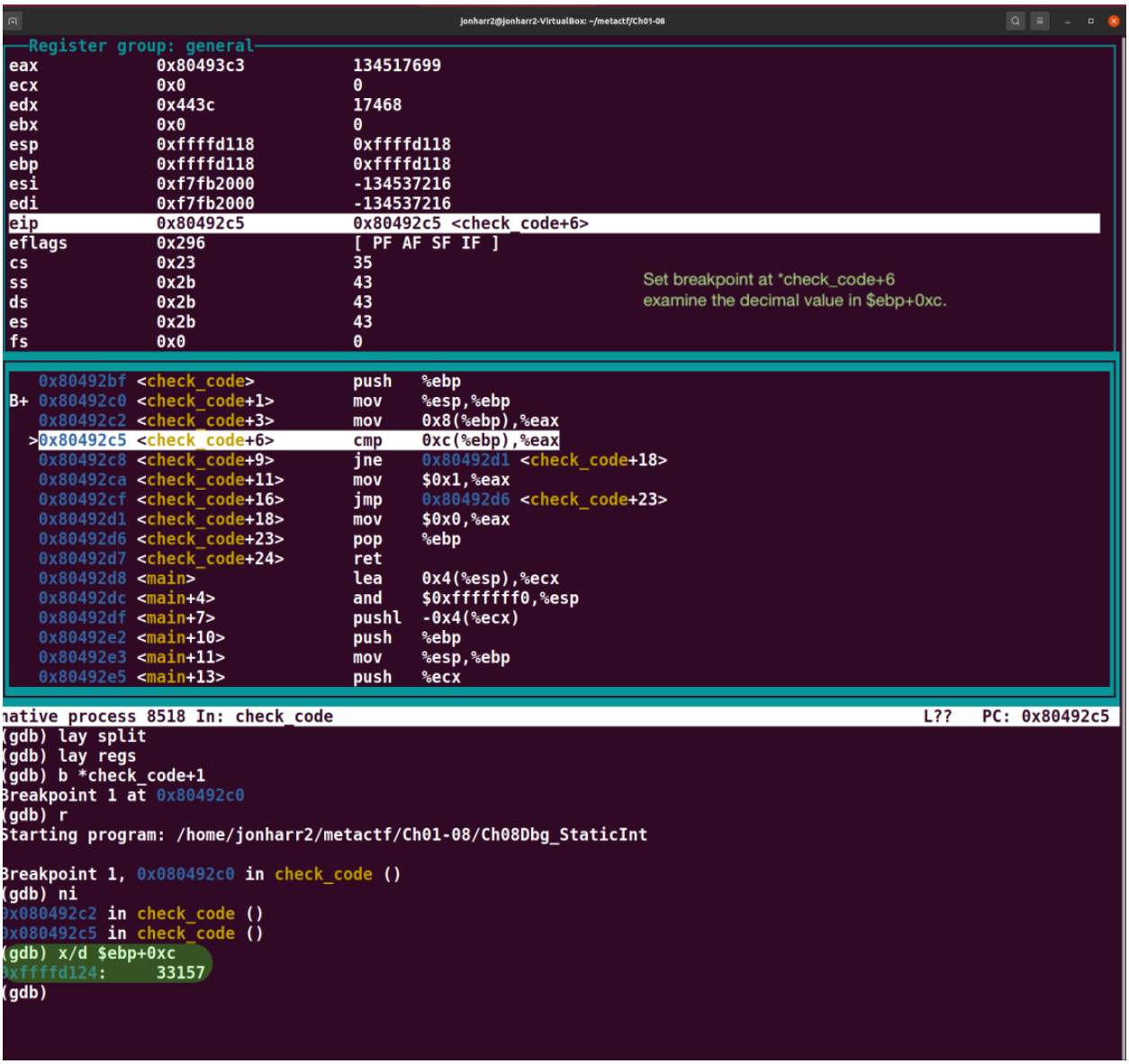
This one is too easy, it tells you right there what the PW is.

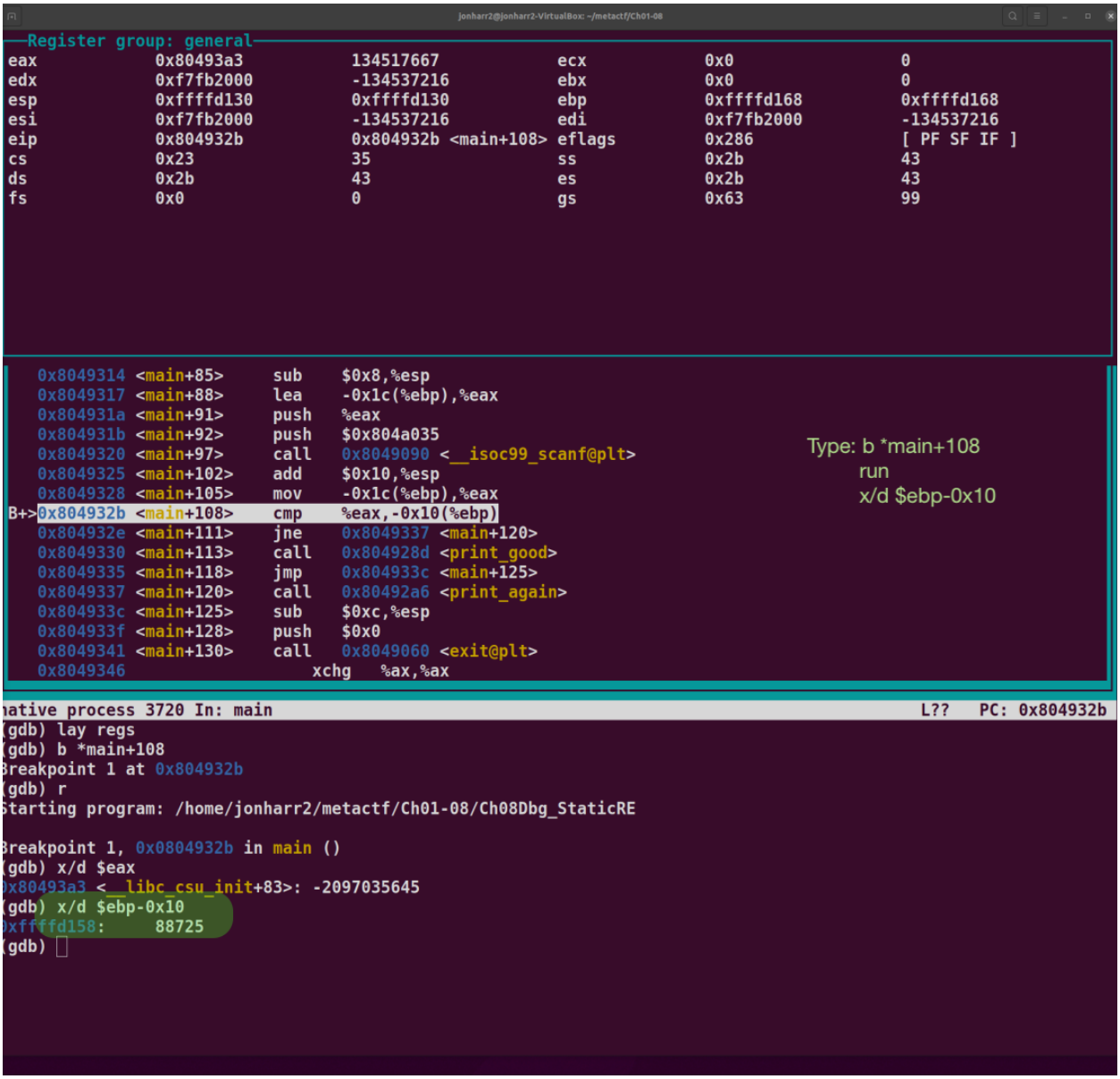
Note: l's and 1's look the same, so if its not working swap them.

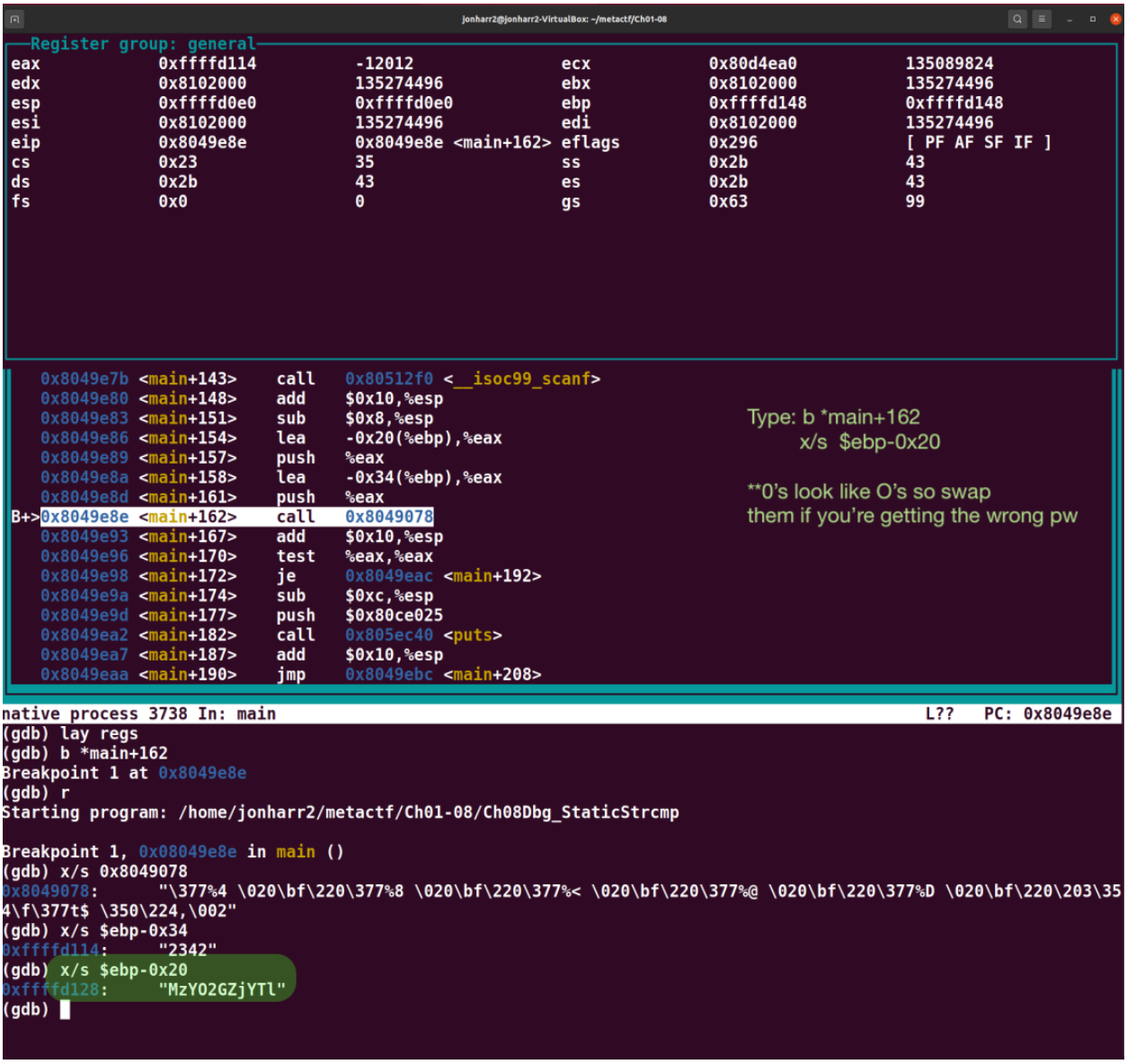

jonharr2@jonharr2-VirtualBox: ~/metactf/Ch01-08
0x804930d <retval_in_rax> push %ebp
0x804930e <retval_in_rax+1> mov %esp,%ebp
0x8049310 <retval_in_rax+3> sub $0x18,%esp
0x8049313 <retval_in_rax+6> sub $0xc,%esp
0x8049316 <retval_in_rax+9> push $0x168d
0x804931b <retval_in_rax+14> call 0x8049090 <srand@plt>
0x8049320 <retval_in_rax+19> add $0x10,%esp
0x8049323 <retval_in_rax+22> movl $0x0,-0x10(%ebp)
0x804932a <retval_in_rax+29> jmp 0x8049338 <retval_in_rax+43>
0x804932c <retval_in_rax+31> call 0x80490c0 <rand@plt>
0x8049331 <retval_in_rax+36> mov %eax,-0xc(%ebp)
0x8049334 <retval_in_rax+39> addl $0x1,-0x10(%ebp)
0x8049338 <retval_in_rax+43> cmpl $0x9,-0x10(%ebp)
0x804933c <retval_in_rax+47> jle 0x804932c <retval_in_rax+31>
0x804933e <retval_in_rax+49> mov -0xc(%ebp),%eax
B+>0x8049341 <retval_in_rax+52> leave
0x8049342 <retval_in_rax+53> ret
0x8049343 <main> lea 0x4(%esp),%ecx
0x8049347 <main+4> and $0xfffffff0,%esp
0x804934a <main+7> pushl -0x4(%ecx)
0x804934d <main+10> push %ebp
0x804934e <main+11> mov %esp,%ebp
0x8049350 <main+13> push %ecx
0x8049351 <main+14> sub $0x14,%esp
0x8049354 <main+17> mov %gs:0x14,%eax
0x804935a <main+23> mov %eax,-0xc(%ebp)
0x804935d <main+26> xor %eax,%eax
0x804935f <main+28> call 0x8049236 <print_msg>
0x8049364 <main+33> sub $0xc,%esp
0x8049367 <main+36> push $0x1
0x8049369 <main+38> call 0x8049050 <sleep@plt>
0x804936e <main+43> add $0x10,%esp
0x8049371 <main+46> sub $0xc,%esp
0x8049374 <main+49> push $0x804a020
0x8049379 <main+54> call 0x8049040 <printf@plt>
Set breakpoint at end of retval_in_rax like so:
b *retval_in_rax+52
Type p $eax
native process 2127 In: retval_in_rax L?? PC: 0x8049341
2 breakpoint keep y 0x08049341 <retval_in_rax+52>
(gdb) d 1
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/jonharr2/metactf/Ch01-08/Ch08Dbg_GdbRegs
Breakpoint 2, 0x08049341 in retval_in_rax ()
(gdb) i r $eax
eax 0x6558a235 1700307509
(gdb) x /s $eax
0x6558a235: <error: Cannot access memory at address 0x6558a235>
(gdb) x /s $ebp-0xc
0xffffd13c: "5\242Xe5\240\004\bT\321\377\377h\321\377\377\232\223\004\b\001"
(gdb) x /s $ebp-0x10
0xffffd138: "\n"
(gdb) p $eax
$1 = 1700307509
(gdb)

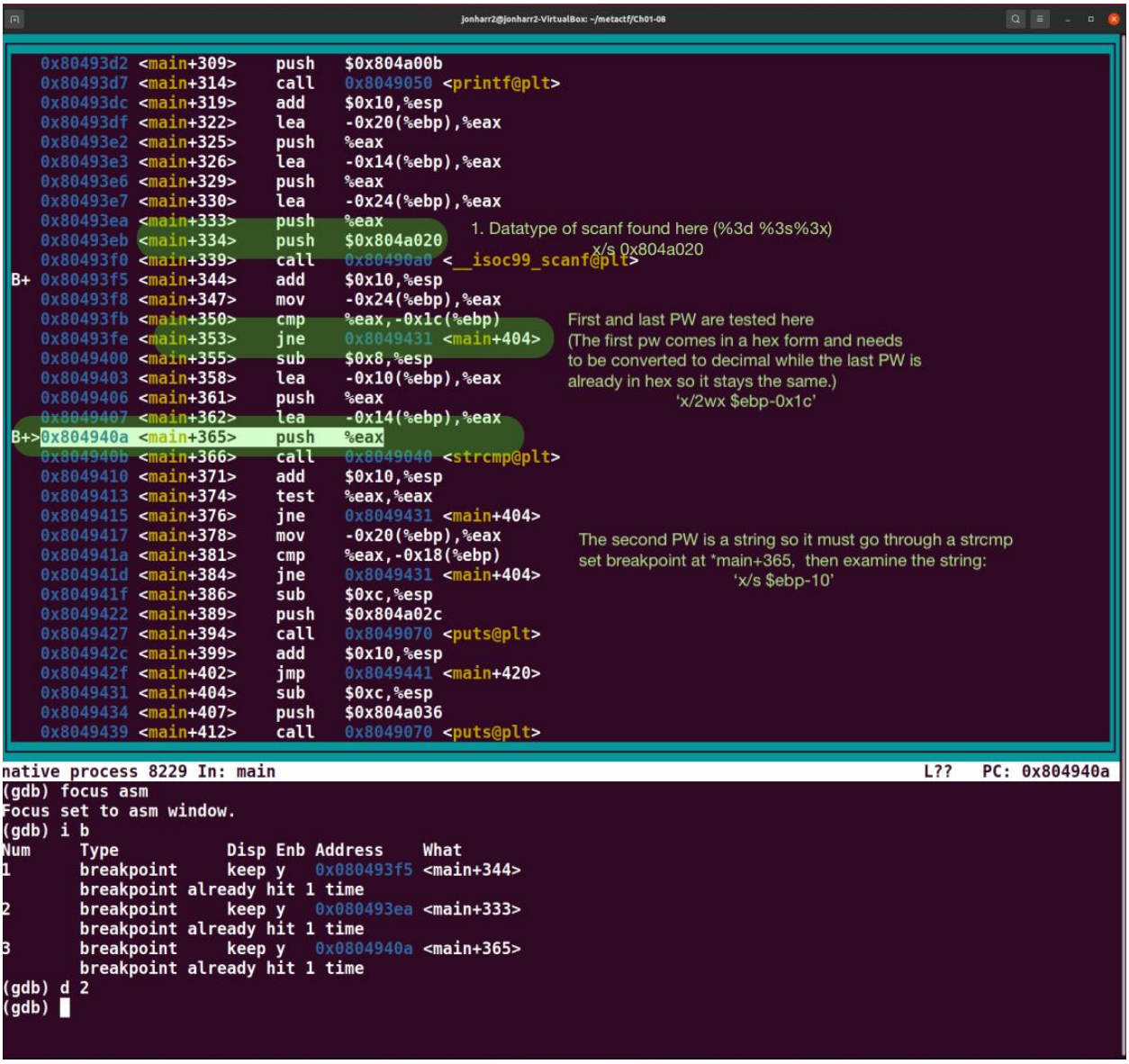

jonharr2@jonharr2-VirtualBox: ~/metactf/Ch01-08
Register group: general
eax 0x1 1
ecx 0x0 0
edx 0xffffd138 -11976
ebx 0x0 0
esp 0xffffd130 0xffffd130
ebp 0xffffd168 0xffffd168
esi 0xf7fb2000 -134537216
edi 0xf7fb2000 -134537216
eip 0x804973d 0x804973d <main+110>
eflags 0x286 [PF SF IF]
cs 0x23 35
ss 0x2b 43
ds 0x2b 43
es 0x2b 43
fs 0x0 0
Set breakpoint at *main+110
type 'set {int*} 0xffffd134 = 0x70
(0xffffd134 is the address for $ebp-0x34)
This will get you in the print_psw function.
0x8049729 <main+90> sub $0x8,%esp
0x804972c <main+93> lea -0x30(%ebp),%eax
0x804972f <main+96> push %eax
0x8049730 <main+97> push $0x804a050
0x8049735 <main+102> call 0x80490b0 <__isoc99_scanf@plt>
0x804973a <main+107> add $0x10,%esp
B+>0x804973d <main+110> cmpl $0x70,-0x34(%ebp)
0x8049741 <main+114> jne 0x8049754 <main+133>
0x8049743 <main+116> sub $0x8,%esp
0x8049746 <main+119> pushl -0x24(%ebp)
0x8049749 <main+122> pushl -0x28(%ebp)
0x804974c <main+125> call 0x804958a <print_pswd>
0x8049751 <main+130> add $0x10,%esp
0x8049754 <main+133> mov -0x30(%ebp),%eax
0x8049757 <main+136> mov -0x2c(%ebp),%edx
native process 8694 In: main L?? PC: 0x804973d
(gdb) lay regs
(gdb) b *main+110
Breakpoint 1 at 0x804973d
(gdb) r
Starting program: /home/jonharr2/metactf/Ch01-08/Ch08Dbg_GdbSetmem
Breakpoint 1, 0x0804973d in main ()
(gdb) x/s $ebp-0x34
0xffffd134: "\001"
(gdb) x/xw $ebp-0x34
0xffffd134: 0x00000001
(gdb) set {int*} 0xffffd134 = 0x70
(gdb) x/xw $ebp-0x34
0xffffd134: 0x00000070
(gdb)

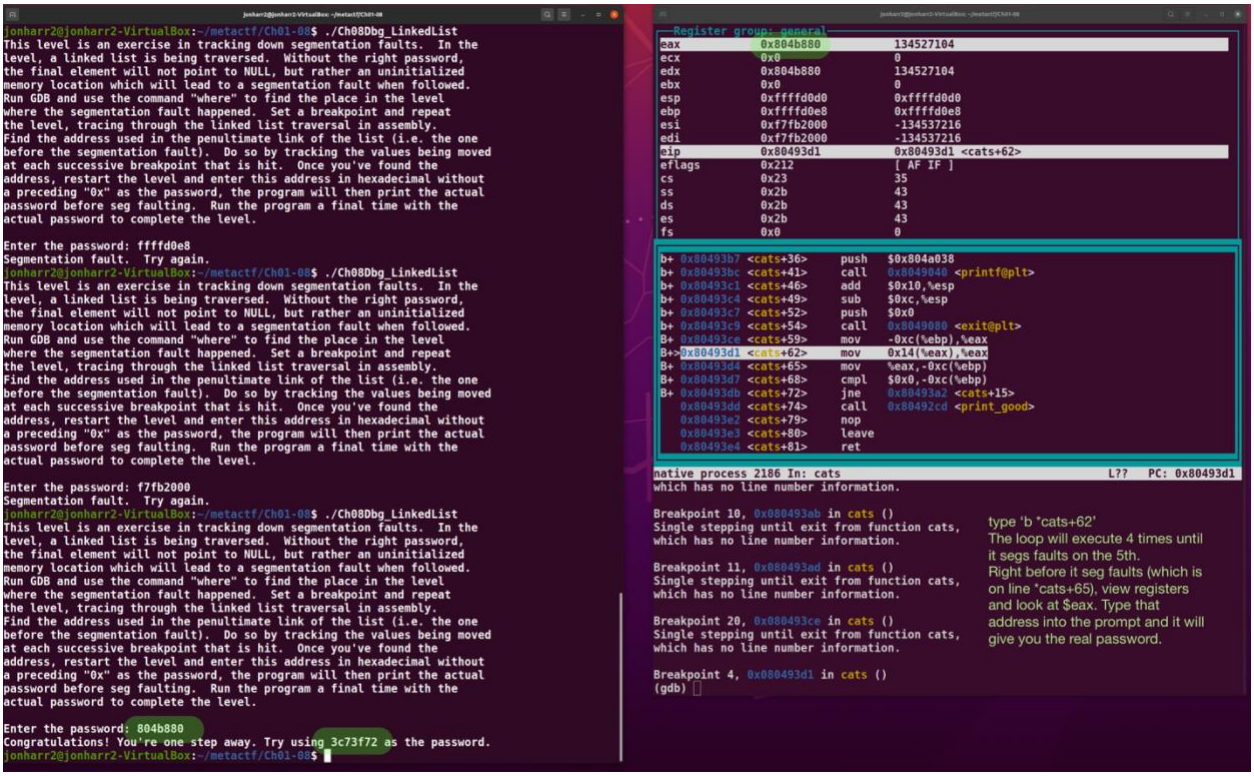

jonharr2@jonharr2-VirtualBox: ~/metactf/Ch01-08
Register group: general
eax 0x4e 78
ecx 0x59 89
edx 0xffffd0fe -12034
ebx 0x0 0
esp 0xffffd070 0xffffd070
ebp 0xffffd098 0xffffd098
esi 0xf7fb2000 -134537216
edi 0xf7fb2000 -134537216
eip 0x804925e 0x804925e <test_c+67>
eflags 0x296 [PF AF SF IF]
cs 0x23 35
Set breakpoint at *test_c+67
x/xw $ebp+0xc gives you address
x/s 0xffffd0fc derefences the string at
the passwords address.
0x8049253 <test_c+56> mov %al,(%edx)
0x8049255 <test_c+58> sub $0x8,%esp
0x8049258 <test_c+61> pushl 0xc(%ebp)
0x804925b <test_c+64> pushl 0x8(%ebp)
B+>0x804925e <test_c+67> call 0x8049040 <strcmp@plt>
0x8049263 <test_c+72> add $0x10,%esp
0x8049266 <test_c+75> test %eax,%eax
0x8049268 <test_c+77> je 0x804927c <test_c+97>
0x804926a <test_c+79> sub $0xc,%esp
0x804926d <test_c+82> push $0x804a00b
0x8049272 <test_c+87> call 0x8049070 <puts@plt>
native process 8816 In: test_c L?? PC: 0x804925e
(gdb) b *test_c+67
Breakpoint 1 at 0x804925e
(gdb) x/xw $ebp+0xc
No registers.
(gdb) p $ebp+0x8
No registers.
(gdb) r
Starting program: /home/jonharr2/metactf/Ch01-08/Ch08Dbg_Radare2Intro2
Breakpoint 1, 0x0804925e in test_c ()
(gdb) x/xw $ebp+0xc
0xffffd0a4: 0xffffd0fc
(gdb) x/s $ebp+0xc
0xffffd0a4: "\374\320\377\377\304\320\377\377"
(gdb) x/s $ebp+0x8
0xffffd0a0: "\350\320\377\377\374\320\377\377\304\320\377\377"
(gdb) x/d $ebp+0x8
0xffffd0a0: -24
(gdb) x/d $ebp+0xc
0xffffd0a4: -4
(gdb) x/s 0xffffd0fc
0xffffd0fc: "YjNkxjUx"
(gdb)


jonharr2@jonharr2-VirtualBox: ~/metactf/Ch01-08
Register group: general
eax 0x80493c3 134517699
ecx 0x0 0
edx 0x443c 17468
ebx 0x0 0
esp 0xffffd118 0xffffd118
ebp 0xffffd118 0xffffd118
esi 0xf7fb2000 -134537216
edi 0xf7fb2000 -134537216
eip 0x80492c5 0x80492c5 <check_code+6>
eflags 0x296 [PF AF SF IF]
cs 0x23 35
ss 0x2b 43
ds 0x2b 43
es 0x2b 43
fs 0x0 0
Set breakpoint at *check_code+6
examine the decimal value in $ebp+0xc.
0x80492bf <check_code> push %ebp
B+ 0x80492c0 <check_code+1> mov %esp,%ebp
0x80492c2 <check_code+3> mov 0x8(%ebp),%eax
>0x80492c5 <check_code+6> cmp 0xc(%ebp),%eax
0x80492c8 <check_code+9> jne 0x80492d1 <check_code+18>
0x80492ca <check_code+11> mov $0x1,%eax
0x80492cf <check_code+16> jmp 0x80492d6 <check_code+23>
0x80492d1 <check_code+18> mov $0x0,%eax
0x80492d6 <check_code+23> pop %ebp
0x80492d7 <check_code+24> ret
0x80492d8 <main> lea 0x4(%esp),%ecx
0x80492dc <main+4> and $0xfffffff0,%esp
0x80492df <main+7> pushl -0x4(%ecx)
0x80492e2 <main+10> push %ebp
0x80492e3 <main+11> mov %esp,%ebp
0x80492e5 <main+13> push %ecx
native process 8518 In: check_code L?? PC: 0x80492c5
(gdb) lay split
(gdb) lay regs
(gdb) b *check_code+1
Breakpoint 1 at 0x80492c0
(gdb) r
Starting program: /home/jonharr2/metactf/Ch01-08/Ch08Dbg_StaticInt
Breakpoint 1, 0x080492c0 in check_code ()
(gdb) ni
0x080492c2 in check_code ()
0x080492c5 in check_code ()
(gdb) x/d $ebp+0xc
0xffffd124: 33157
(gdb)


jonharr2@jonharr2-VirtualBox: ~/metactf/Ch01-08
Register group: general
eax 0x80493a3 134517667 ecx 0x0 0
edx 0xf7fb2000 -134537216 ebx 0x0 0
esp 0xffffd130 0xffffd130 ebp 0xffffd168 0xffffd168
esi 0xf7fb2000 -134537216 edi 0xf7fb2000 -134537216
eip 0x804932b 0x804932b <main+108> eflags 0x286 [PF SF IF]
cs 0x23 35 ss 0x2b 43
ds 0x2b 43 es 0x2b 43
fs 0x0 0 gs 0x63 99
0x8049314 <main+85> sub $0x8,%esp
0x8049317 <main+88> lea -0x1c(%ebp),%eax
0x804931a <main+91> push %eax
0x804931b <main+92> push $0x804a035
0x8049320 <main+97> call 0x8049090 <__isoc99_scanf@plt>
0x8049325 <main+102> add $0x10,%esp
0x8049328 <main+105> mov -0x1c(%ebp),%eax
B+>0x804932b <main+108> cmp %eax,-0x10(%ebp)
0x804932e <main+111> jne 0x8049337 <main+120>
0x8049330 <main+113> call 0x804928d <print_good>
0x8049335 <main+118> jmp 0x804933c <main+125>
0x8049337 <main+120> call 0x80492a6 <print_again>
0x804933c <main+125> sub $0xc,%esp
0x804933f <main+128> push $0x0
0x8049341 <main+130> call 0x8049060 <exit@plt>
0x8049346 xchg %ax,%ax
Type: b *main+108
run
x/d $ebp-0x10
native process 3720 In: main L?? PC: 0x804932b
(gdb) lay regs
(gdb) b *main+108
Breakpoint 1 at 0x804932b
(gdb) r
Starting program: /home/jonharr2/metactf/Ch01-08/Ch08Dbg_StaticRE
Breakpoint 1, 0x0804932b in main ()
(gdb) x/d $eax
0x80493a3 <__libc_csu_init+83>: -2097035645
(gdb) x/d $ebp-0x10
0xffffd158: 88725
(gdb)


jonharr2@jonharr2-VirtualBox: ~/metactf/Ch01-08
Register group: general
eax 0xffffd114 -12012 ecx 0x80d4ea0 135089824
edx 0x8102000 135274496 ebx 0x8102000 135274496
esp 0xffffd0e0 0xffffd0e0 ebp 0xffffd148 0xffffd148
esi 0x8102000 135274496 edi 0x8102000 135274496
eip 0x8049e8e 0x8049e8e <main+162> eflags 0x296 [PF AF SF IF]
cs 0x23 35 ss 0x2b 43
ds 0x2b 43 es 0x2b 43
fs 0x0 0 gs 0x63 99
0x8049e7b <main+143> call 0x80512f0 <__isoc99_scanf>
0x8049e80 <main+148> add $0x10,%esp
0x8049e83 <main+151> sub $0x8,%esp
0x8049e86 <main+154> lea -0x20(%ebp),%eax
0x8049e89 <main+157> push %eax
0x8049e8a <main+158> lea -0x34(%ebp),%eax
0x8049e8d <main+161> push %eax
B+>0x8049e8e <main+162> call 0x8049078
0x8049e93 <main+167> add $0x10,%esp
0x8049e96 <main+170> test %eax,%eax
0x8049e98 <main+172> je 0x8049eac <main+192>
0x8049e9a <main+174> sub $0xc,%esp
0x8049e9d <main+177> push $0x80ce025
0x8049ea2 <main+182> call 0x805ec40 <puts>
0x8049ea7 <main+187> add $0x10,%esp
0x8049eaa <main+190> jmp 0x8049ebc <main+208>
Type: b *main+162
x/s $ebp-0x20
**0's look like O's so swap
them if you're getting the wrong pw
native process 3738 In: main L?? PC: 0x8049e8e
(gdb) lay regs
(gdb) b *main+162
Breakpoint 1 at 0x8049e8e
(gdb) r
Starting program: /home/jonharr2/metactf/Ch01-08/Ch08Dbg_StaticStrcmp
Breakpoint 1, 0x08049e8e in main ()
(gdb) x/s 0x8049078
0x8049078: "\377%4 \020\bf\220\377%8 \020\bf\220\377%< \020\bf\220\377%@ \020\bf\220\377%D \020\bf\220\203\35
4\f\377t$ \350\224,\002"
(gdb) x/s $ebp-0x34
0xffffd114: "2342"
(gdb) x/s $ebp-0x20
0xffffd128: "MzYO2GZjYTl"
(gdb)


jonharr2@jonharr2-VirtualBox: ~/metactf/Ch01-08
0x80493d2 <main+309> push $0x804a00b
0x80493d7 <main+314> call 0x8049050 <printf@plt>
0x80493dc <main+319> add $0x10,%esp
0x80493df <main+322> lea -0x20(%ebp),%eax
0x80493e2 <main+325> push %eax
0x80493e3 <main+326> lea -0x14(%ebp),%eax
0x80493e6 <main+329> push %eax
0x80493e7 <main+330> lea -0x24(%ebp),%eax
0x80493ea <main+333> push %eax
0x80493eb <main+334> push $0x804a020
0x80493f0 <main+339> call 0x80490a0 <__isoc99_scanf@plt>
B+ 0x80493f5 <main+344> add $0x10,%esp
0x80493f8 <main+347> mov -0x24(%ebp),%eax
0x80493fb <main+350> cmp %eax,-0x1c(%ebp)
0x80493fe <main+353> jne 0x8049431 <main+404>
0x8049400 <main+355> sub $0x8,%esp
0x8049403 <main+358> lea -0x10(%ebp),%eax
0x8049406 <main+361> push %eax
0x8049407 <main+362> lea -0x14(%ebp),%eax
B+>0x804940a <main+365> push %eax
0x804940b <main+366> call 0x8049040 <strcmp@plt>
0x8049410 <main+371> add $0x10,%esp
0x8049413 <main+374> test %eax,%eax
0x8049415 <main+376> jne 0x8049431 <main+404>
0x8049417 <main+378> mov -0x20(%ebp),%eax
0x804941a <main+381> cmp %eax,-0x18(%ebp)
0x804941d <main+384> jne 0x8049431 <main+404>
0x804941f <main+386> sub $0xc,%esp
0x8049422 <main+389> push $0x804a02c
0x8049427 <main+394> call 0x8049070 <puts@plt>
0x804942c <main+399> add $0x10,%esp
0x804942f <main+402> jmp 0x8049441 <main+420>
0x8049431 <main+404> sub $0xc,%esp
0x8049434 <main+407> push $0x804a036
0x8049439 <main+412> call 0x8049070 <puts@plt>
1. Datatype of scanf found here (%3d %3s%3x)
x/s 0x804a020
First and last PW are tested here
(The first pw comes in a hex form and needs
to be converted to decimal while the last PW is
already in hex so it stays the same.)
'x/2wx $ebp-0x1c'
The second PW is a string so it must go through a strcmp
set breakpoint at *main+365, then examine the string:
'x/s $ebp-10'
native process 8229 In: main L?? PC: 0x804940a
(gdb) focus asm
Focus set to asm window.
(gdb) i b
Num Type Disp Enb Address What
1 breakpoint keep y 0x080493f5 <main+344>
breakpoint already hit 1 time
2 breakpoint keep y 0x080493ea <main+333>
breakpoint already hit 1 time
3 breakpoint keep y 0x0804940a <main+365>
breakpoint already hit 1 time
(gdb) d 2
(gdb)


jonharr2@jonharr2-VirtualBox:~/metactf/Ch01-08$ ./Ch08Dbg_LinkedList
This level is an exercise in tracking down segmentation faults. In the level, a linked list is being traversed. Without the right password, the final element will not point to NULL, but rather an uninitialized memory location which will lead to a segmentation fault when followed. Run GDB and use the command "where" to find the place in the level where the segmentation fault happened. Set a breakpoint and repeat the level, tracing through the linked list traversal in assembly. Find the address used in the penultimate link of the list (i.e. the one before the segmentation fault). Do so by tracking the values being moved at each successive breakpoint that is hit. Once you've found the address, restart the level and enter this address in hexadecimal without a preceding "0x" as the password, the program will then print the actual password before seg faulting. Run the program a final time with the actual password to complete the level.
Enter the password: ffffd0e8
Segmentation fault. Try again.
jonharr2@jonharr2-VirtualBox:~/metactf/Ch01-08$ ./Ch08Dbg_LinkedList
This level is an exercise in tracking down segmentation faults. In the level, a linked list is being traversed. Without the right password, the final element will not point to NULL, but rather an uninitialized memory location which will lead to a segmentation fault when followed. Run GDB and use the command "where" to find the place in the level where the segmentation fault happened. Set a breakpoint and repeat the level, tracing through the linked list traversal in assembly. Find the address used in the penultimate link of the list (i.e. the one before the segmentation fault). Do so by tracking the values being moved at each successive breakpoint that is hit. Once you've found the address, restart the level and enter this address in hexadecimal without a preceding "0x" as the password, the program will then print the actual password before seg faulting. Run the program a final time with the actual password to complete the level.
Enter the password: f7fb2000
Segmentation fault. Try again.
jonharr2@jonharr2-VirtualBox:~/metactf/Ch01-08$ ./Ch08Dbg_LinkedList
This level is an exercise in tracking down segmentation faults. In the level, a linked list is being traversed. Without the right password, the final element will not point to NULL, but rather an uninitialized memory location which will lead to a segmentation fault when followed. Run GDB and use the command "where" to find the place in the level where the segmentation fault happened. Set a breakpoint and repeat the level, tracing through the linked list traversal in assembly. Find the address used in the penultimate link of the list (i.e. the one before the segmentation fault). Do so by tracking the values being moved at each successive breakpoint that is hit. Once you've found the address, restart the level and enter this address in hexadecimal without a preceding "0x" as the password, the program will then print the actual password before seg faulting. Run the program a final time with the actual password to complete the level.
Enter the password: 804b880
Congratulations! You're one step away. Try using 3c73f72 as the password.
jonharr2@jonharr2-VirtualBox:~/metactf/Ch01-08$
Register group: general
eax 0x804b880 134527104
ecx 0x0 0
edx 0x804b880 134527104
ebx 0x0 0
esp 0xffffd0d0 0xffffd0d0
ebp 0xffffd0e8 0xffffd0e8
esi 0xf7fb2000 -134537216
edi 0xf7fb2000 -134537216
eip 0x80493d1 0x80493d1 <cats+62>
eflags 0x212 [AF IF]
cs 0x23 35
ss 0x2b 43
ds 0x2b 43
es 0x2b 43
fs 0x0 0
b+ 0x80493b7 <cats+36> push $0x804a038
b+ 0x80493bc <cats+41> call 0x8049040 <printf@plt>
b+ 0x80493c1 <cats+46> add $0x10,%esp
b+ 0x80493c4 <cats+49> sub $0xc,%esp
b+ 0x80493c7 <cats+52> push $0x0
b+ 0x80493c9 <cats+54> call 0x8049080 <exit@plt>
B+ 0x80493ce <cats+59> mov -0xc(%ebp),%eax
B+>0x80493d1 <cats+62> mov 0x14(%eax),%eax
B+ 0x80493d4 <cats+65> mov %eax,-0xc(%ebp)
B+ 0x80493d7 <cats+68> cmpl $0x0,-0xc(%ebp)
B+ 0x80493db <cats+72> jne 0x80493a2 <cats+15>
0x80493dd <cats+74> call 0x80492cd <print_good>
0x80493e2 <cats+79> nop
0x80493e3 <cats+80> leave
0x80493e4 <cats+81> ret
native process 2186 In: cats L?? PC: 0x80493d1
which has no line number information.
Breakpoint 10, 0x080493ab in cats ()
Single stepping until exit from function cats, which has no line number information.
Breakpoint 11, 0x080493ad in cats ()
Single stepping until exit from function cats, which has no line number information.
Breakpoint 20, 0x080493ce in cats ()
Single stepping until exit from function cats, which has no line number information.
Breakpoint 4, 0x080493d1 in cats ()
(gdb)
type 'b *cats+62'
The loop will execute 4 times until it segs faults on the 5th.
Right before it seg faults (which is on line *cats+65), view registers and look at $eax. Type that address into the prompt and it will give you the real password.

Objdump -d Ch11MalBeh_HijackPLT

<print_good> is towards the top

<sleep> is towards the bottom

```
59575b96 <print_good>:
59575b96:       55                      push   %ebp
59575b97:       89 e5                   mov    %esp,%ebp
59575b99:       83 ec 08                sub    $0x8,%esp
59575b9c:       83 ec 0c                sub    $0xc,%esp
59575b9f:       68 08 60 57 59          push   $0x59576008
59575ba4:       e8 c7 34 ad ae          call   8049070 <puts@plt>
59575ba9:       83 c4 10                add    $0x10,%esp
59575bac:       83 ec 0c                sub    $0xc,%esp
59575baf:       6a 00                   push   $0x0
59575bb1:       e8 ca 34 ad ae          call   8049080 <exit@plt>
```

```
jonharr2@jonharr2-VirtualBox: ~/metactf/Ch11-13

Disassembly of section .plt:

08049030 <.plt>:
 8049030:       ff 35 04 80 57 59       pushl  0x59578004
 8049036:       ff 25 08 80 57 59       jmp    *0x59578008
 804903c:       00 00                   add    %al,(%eax)
        ...

08049040 <printf@plt>:
 8049040:       ff 25 0c 80 57 59       jmp    *0x5957800c
 8049046:       68 00 00 00 00          push   $0x0
 804904b:       e9 e0 ff ff ff          jmp    8049030 <.plt>

08049050 <signal@plt>:
 8049050:       ff 25 10 80 57 59       jmp    *0x59578010
 8049056:       68 08 00 00 00          push   $0x8
 804905b:       e9 d0 ff ff ff          jmp    8049030 <.plt>

08049060 <sleep@plt>:
 8049060:       ff 25 14 80 57 59       jmp    *0x59578014
 8049066:       68 10 00 00 00          push   $0x10
 804906b:       e9 c0 ff ff ff          jmp    8049030 <.plt>
```

Password in this case is:

59578014 59575b96

- Run the program, find the UID hint

```
[redacted]~/metactf/Ch11-13$ ./Ch11MalBeh_LdPreloadGetUID
Malware can hijack many functions through the use of LD_PRELOAD. Security
checks using the function getuid() are particularly vulnerable.
In this level, when you guess the password incorrectly, the program will
give you a hint as to what the password is, if you have a specific
UID. If you are able to to hijack the getuid() function and get it to
output a specific UID, you will receive the password. In order to
solve this level, create a dynamic shared object file that implements
getuid() and returns the UID. Use LD_PRELOAD to preload your library.
You will need to employ specific flags within gcc that will produce
dynamic, position-independent, shared object code. Be sure to compile with
your processor's architecture in mind (e.g. -m32 for 32-bit).

Enter the password: 982348982
If you run this program with the UID of 734562, the password will be on the next
line.
```

- Create file: 'vi myGetUID.c', put this inside it with the UID it provides you. Save the file.

```
jonharr2@jonharr2-VirtualBox: ~/metactf/Ch11-13
#include <stdio.h>

int getuid(){
        return 734562;
}
```

- Next, type: gcc -m32 -fPIC -shared -o myGetUID.so myGetUID.c
- Type ls, make sure you have a newly created myGetUID.so file
- Next, type: LD_PRELOAD=./myGetUID.so ./Ch11MalBeh_LdPreloadGetUID

```
jonharr2@jonharr2-VirtualBox:~/metactf/Ch11-13$ LD_PRELOAD=./myGetUID.so ./Ch11M
alBeh_LdPreloadGetUID
Malware can hijack many functions through the use of LD_PRELOAD. Security
checks using the function getuid() are particularly vulnerable.
In this level, when you guess the password incorrectly, the program will
give you a hint as to what the password is, if you have a specific
UID. If you are able to to hijack the getuid() function and get it to
output a specific UID, you will receive the password. In order to
solve this level, create a dynamic shared object file that implements
getuid() and returns the UID. Use LD_PRELOAD to preload your library.
You will need to employ specific flags within gcc that will produce
dynamic, position-independent, shared object code. Be sure to compile with
your processor's architecture in mind (e.g. -m32 for 32-bit).

Enter the password: 734562
If you run this program with the UID of 734562, the password will be on the next
line.

Hint: ESVmRkvG0
Try again.
```

- Create a .c file: 'vi myRand.c' with the following. Save the file

```
int rand(){
return 0;
}
```

- Next, type: 'gcc -m32 -fPIC -shared -o myRand.so myRand.c'
- Type ls to make sure you have a newly created myRand.so file
- Next type: 'LD_PRELOAD=./myRand.so ./Ch11MalBeh_LdPreloadRand'

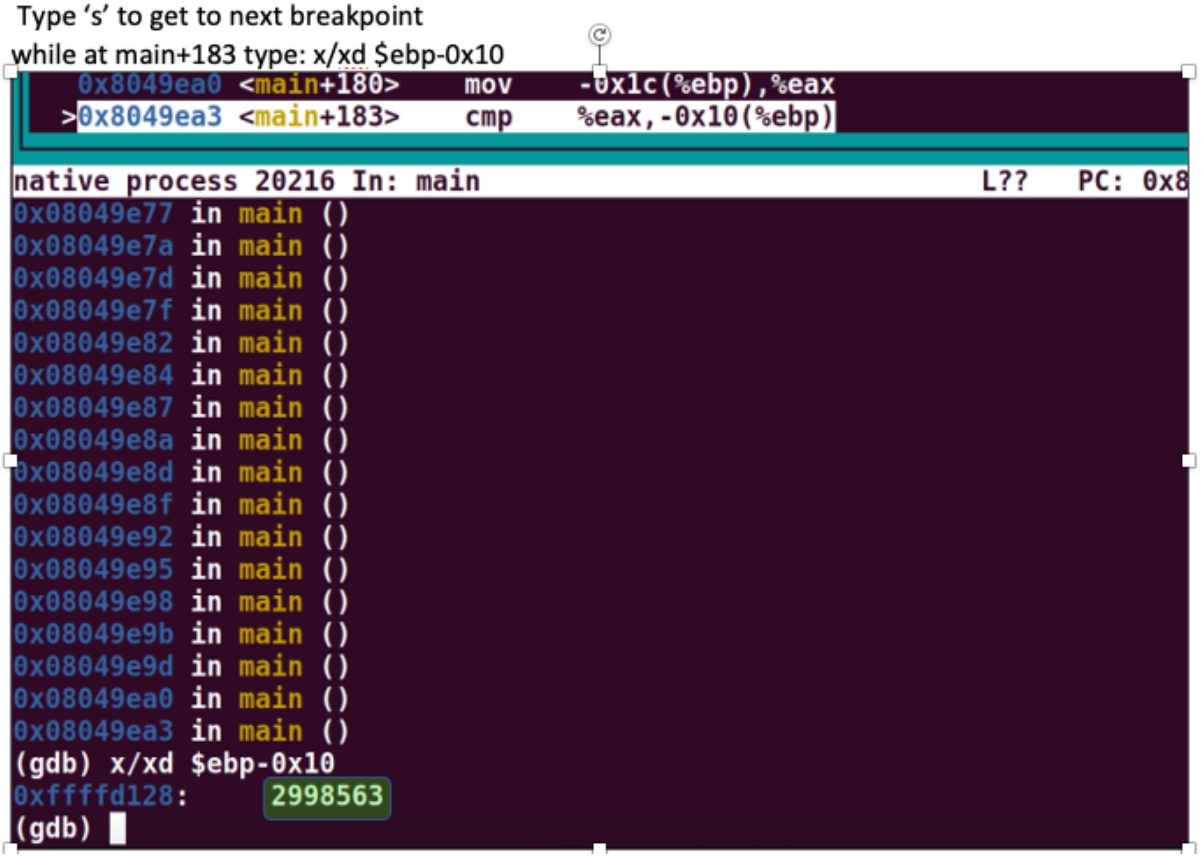
```
~/metactf/Ch11-13$ LD_PRELOAD=./myRand.so ./Ch11Mal
Beh_LdPreloadRand
On Linux, the environment variable LD_PRELOAD is used to specify shared
libraries that should be pre-loaded before attempting to load other dynamic
libraries.  Malware can use this mechanism to preload rogue versions of
standard library calls in order to hijack the execution of these calls.
In this level, when you guess the password incorrectly, the program uses
calls to the rand() function to give you a hint as to what the password is.
If you are able to hijack the rand() function and have it return 0 all of
time, the output you will see will be the password itself.  To solve this
level, create a shared object rand.so that implements rand() by returning
a constant 0.  Ensure that you have gcc generate a shared, 32-bit,
position-independent (-fPIC) object file and force the binary to preload it
to reveal the password.  Note that, while you solve this level in many
ways, the library hijacking method will likely be the easiet.

Enter the password: 23
If you got me to print three zeros in the next line, then the hint is the passwo
rd
0 0 0
Hint: Sm0XUCqi3
```

- Output should produce three zeros

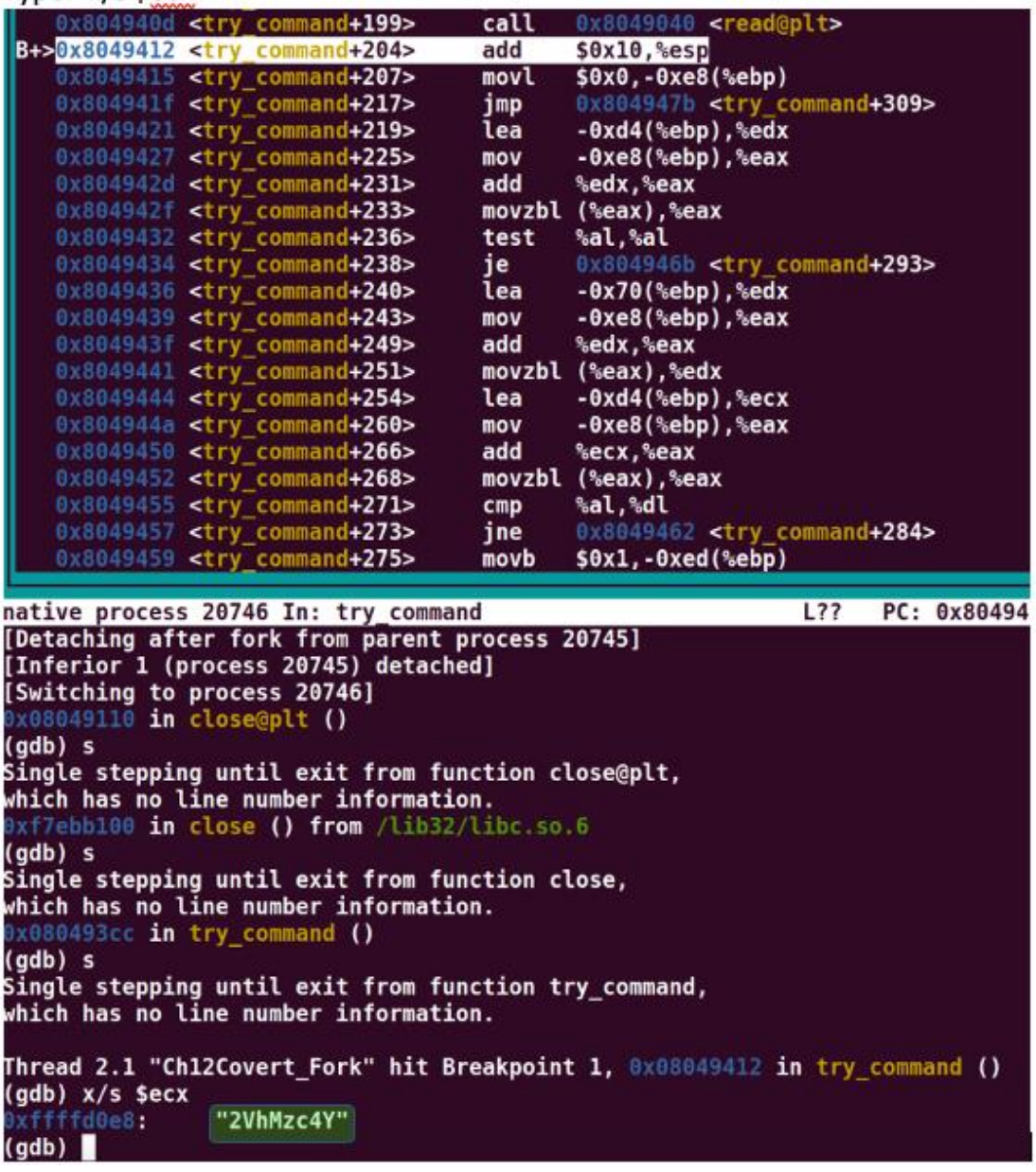
- Open two windows, in one window run the command 'netcat -l 8080'
- In the other window run the program ./Ch11MalBeh_NetcatShovel
- Password should show up on the netcat window

```
~/metactf/Ch11-13$ netcat -l 8080
NDM3ZTNm
```

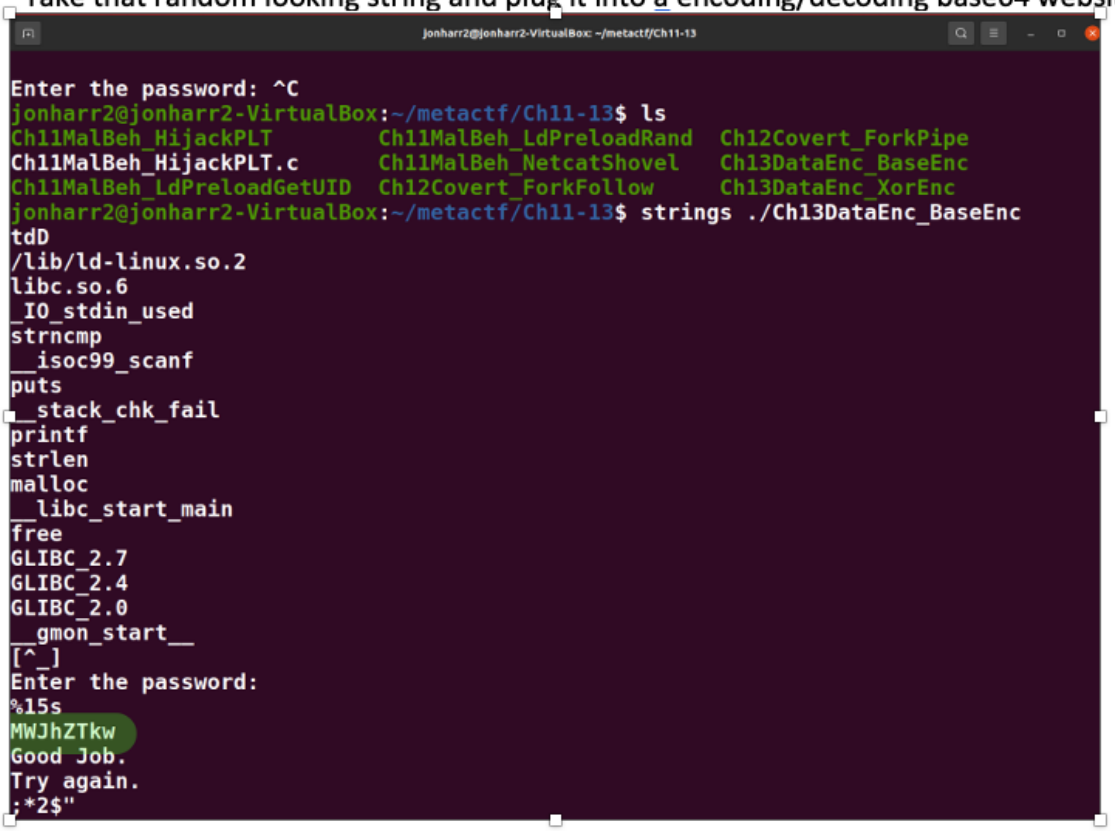
- Open gdb ./Ch12Convert_ForkFollow, lay asm
- Set two breakpoints: *b main+33 and b *main+183
- Run the program
- While at main+33 type: set follow-fork-mode child
- Type ‘s’ to get to next breakpoint
- while at main+183 type: x/xd $ebp-0x10

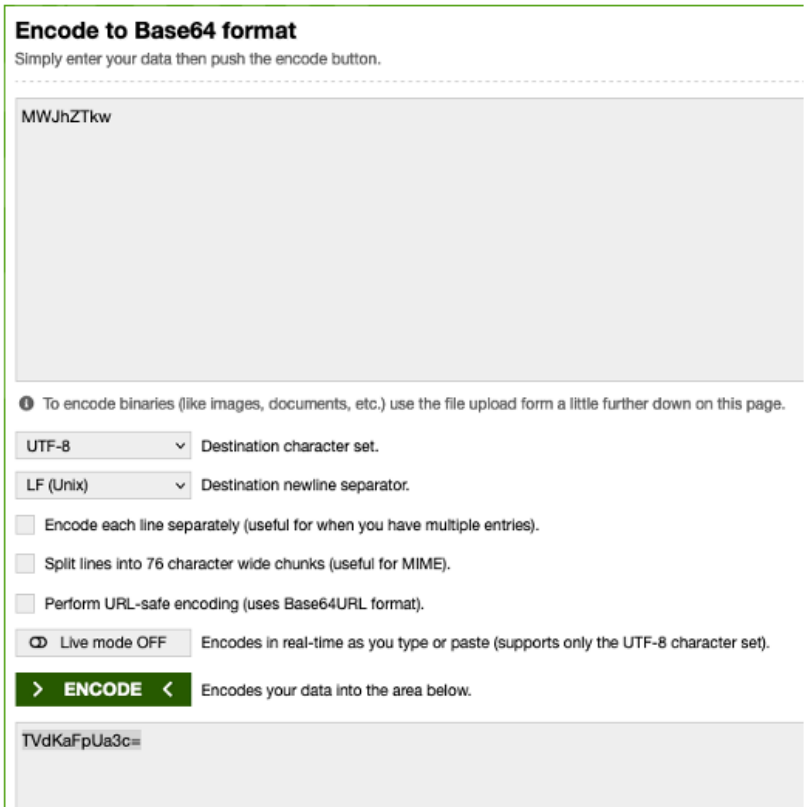


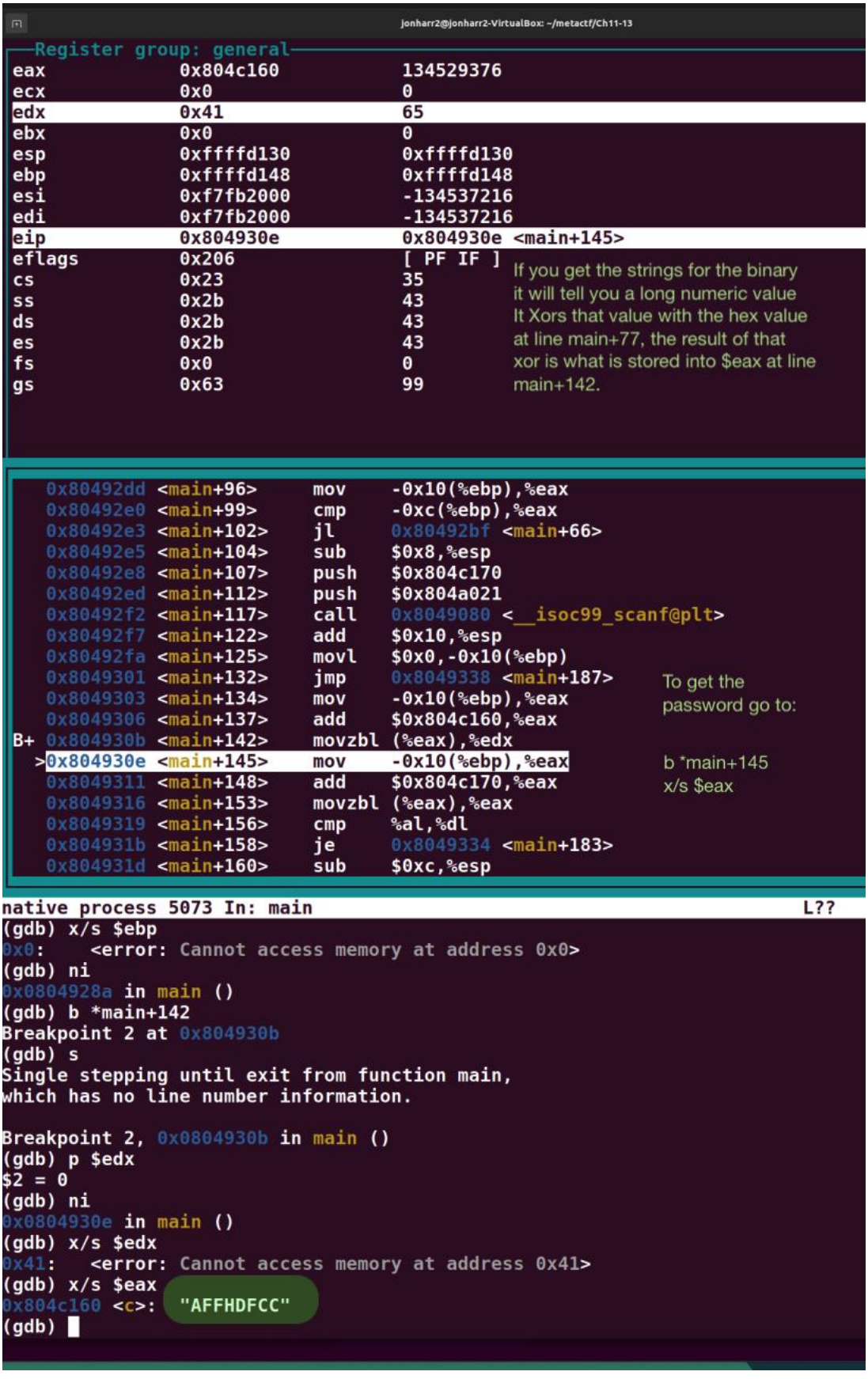
- Open gdb ./Ch12Covert_ForkPipe
- Type:
  - lay asm
  - b *try_command+40
  - b *try_command+204
- Run
- Once you are at the breakpoint (try_command+40) type: 'set follow-fork-mode child'
- Step till you get to <try_command+204>
- Type: 'x/s $ecx'



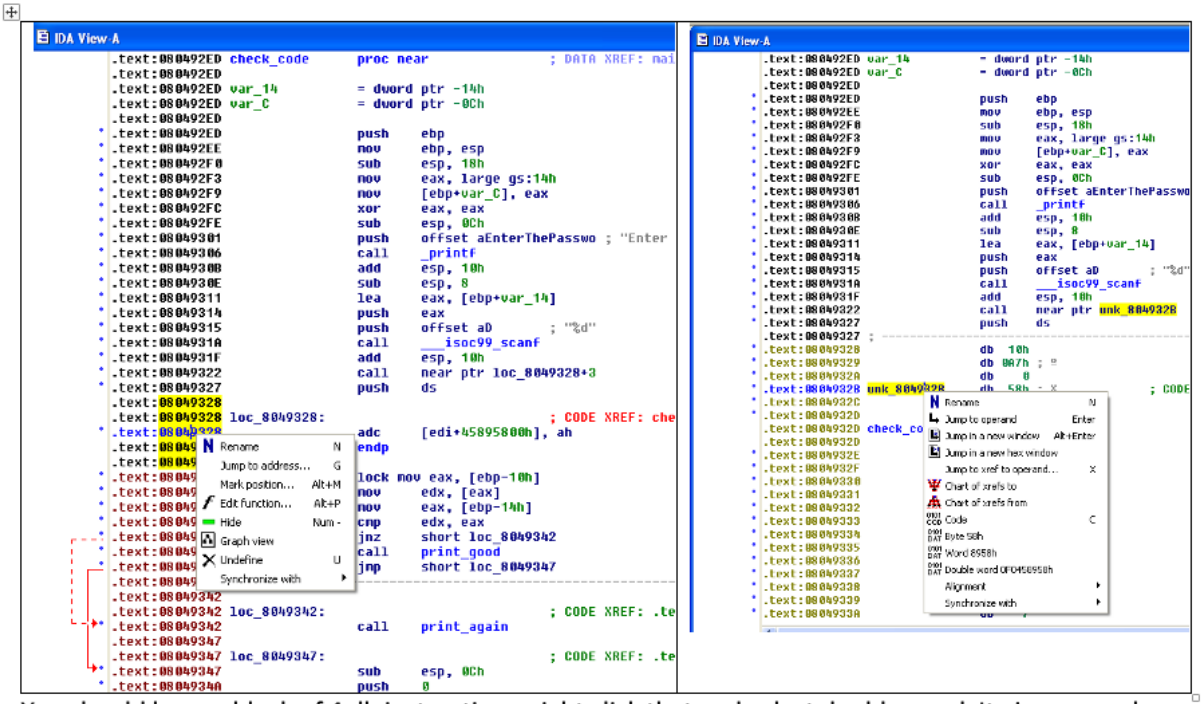
- Run strings on ./Ch13DataEnc_BaseEnc
- Take that random looking string and plug it into a encoding/decoding base64 website



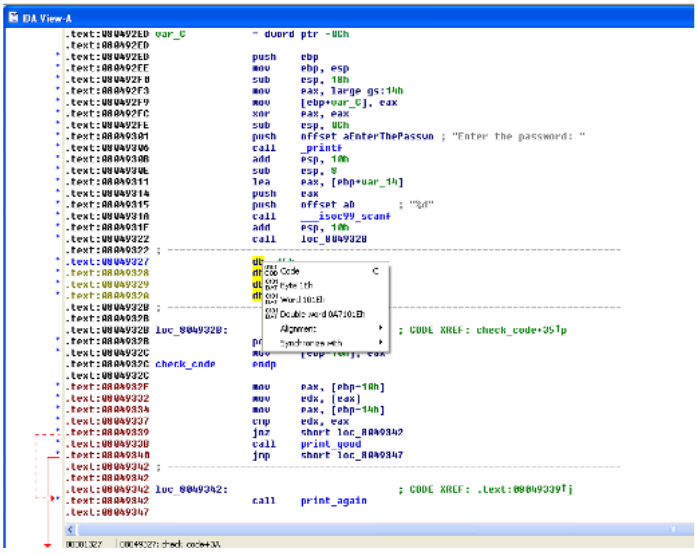



jonharr2@jonharr2-VirtualBox: ~/metactf/Ch11-13
Register group: general
eax 0x804c160 134529376
ecx 0x0 0
edx 0x41 65
ebx 0x0 0
esp 0xffffd130 0xffffd130
ebp 0xffffd148 0xffffd148
esi 0xf7fb2000 -134537216
edi 0xf7fb2000 -134537216
eip 0x804930e 0x804930e <main+145>
eflags 0x206 [PF IF]
cs 0x23 35
ss 0x2b 43
ds 0x2b 43
es 0x2b 43
fs 0x0 0
gs 0x63 99
If you get the strings for the binary it will tell you a long numeric value It Xors that value with the hex value at line main+77, the result of that xor is what is stored into $eax at line main+142.
0x80492dd <main+96> mov -0x10(%ebp),%eax
0x80492e0 <main+99> cmp -0xc(%ebp),%eax
0x80492e3 <main+102> jl 0x80492bf <main+66>
0x80492e5 <main+104> sub $0x8,%esp
0x80492e8 <main+107> push $0x804c170
0x80492ed <main+112> push $0x804a021
0x80492f2 <main+117> call 0x8049080 <__isoc99_scanf@plt>
0x80492f7 <main+122> add $0x10,%esp
0x80492fa <main+125> movl $0x0,-0x10(%ebp)
0x8049301 <main+132> jmp 0x8049338 <main+187>
0x8049303 <main+134> mov -0x10(%ebp),%eax
0x8049306 <main+137> add $0x804c160,%eax
B+ 0x804930b <main+142> movzbl (%eax),%edx
>0x804930e <main+145> mov -0x10(%ebp),%eax
0x8049311 <main+148> add $0x804c170,%eax
0x8049316 <main+153> movzbl (%eax),%eax
0x8049319 <main+156> cmp %al,%dl
0x804931b <main+158> je 0x8049334 <main+183>
0x804931d <main+160> sub $0xc,%esp
To get the password go to:
b *main+145
x/s $eax
native process 5073 In: main L??
(gdb) x/s $ebp
0x0: <error: Cannot access memory at address 0x0>
(gdb) ni
0x0804928a in main ()
(gdb) b *main+142
Breakpoint 2 at 0x804930b
(gdb) s
Single stepping until exit from function main,
which has no line number information.
Breakpoint 2, 0x0804930b in main ()
(gdb) p $edx
$2 = 0
(gdb) ni
0x0804930e in main ()
(gdb) x/s $edx
0x41: <error: Cannot access memory at address 0x41>
(gdb) x/s $eax
0x804c160 <c>: "AFFHDFCC"
(gdb)

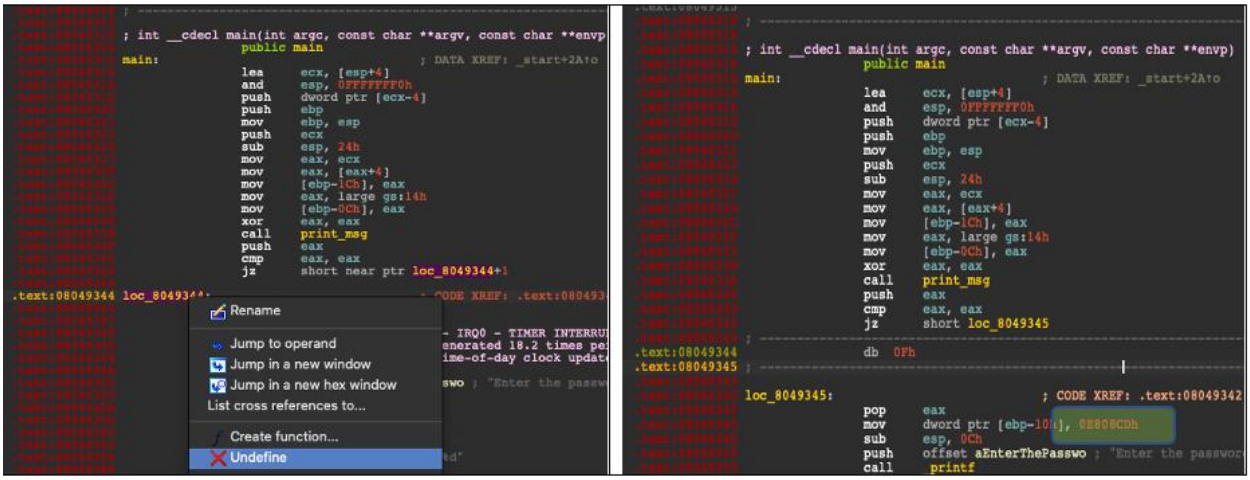
- Open binary in IDA Pro
- Scroll up to Check_code function
- Highlight the block of code that includes the function call loc_8049328 in my case
- Click undefine or press U
- Redefine the function (unk_804932b) as code
- Undefine the push instruction @ text:08049327



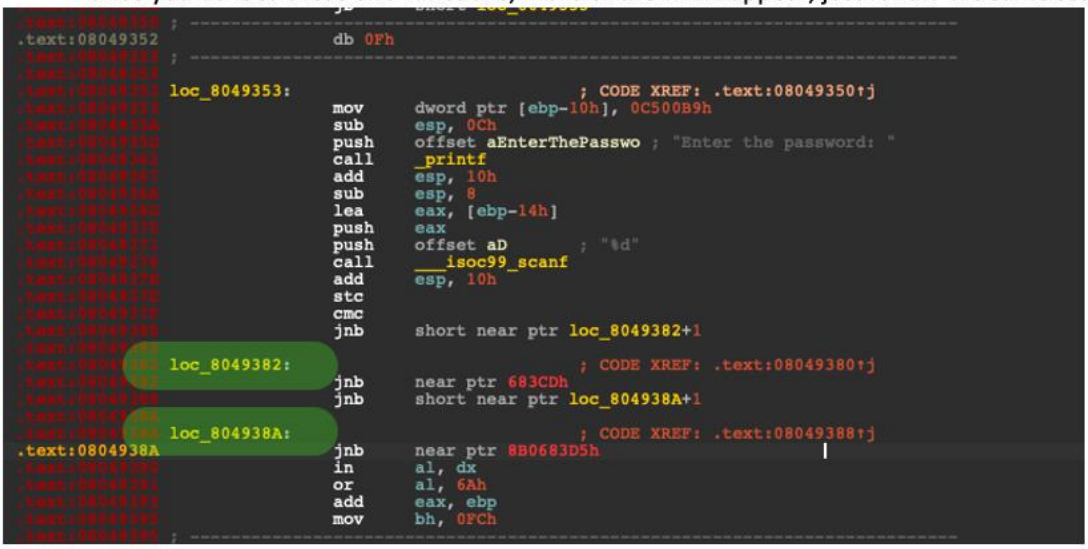
You should have a block of 4 db instructions, right click that and select double word, It gives you a hex value, convert that hex value and that is your password



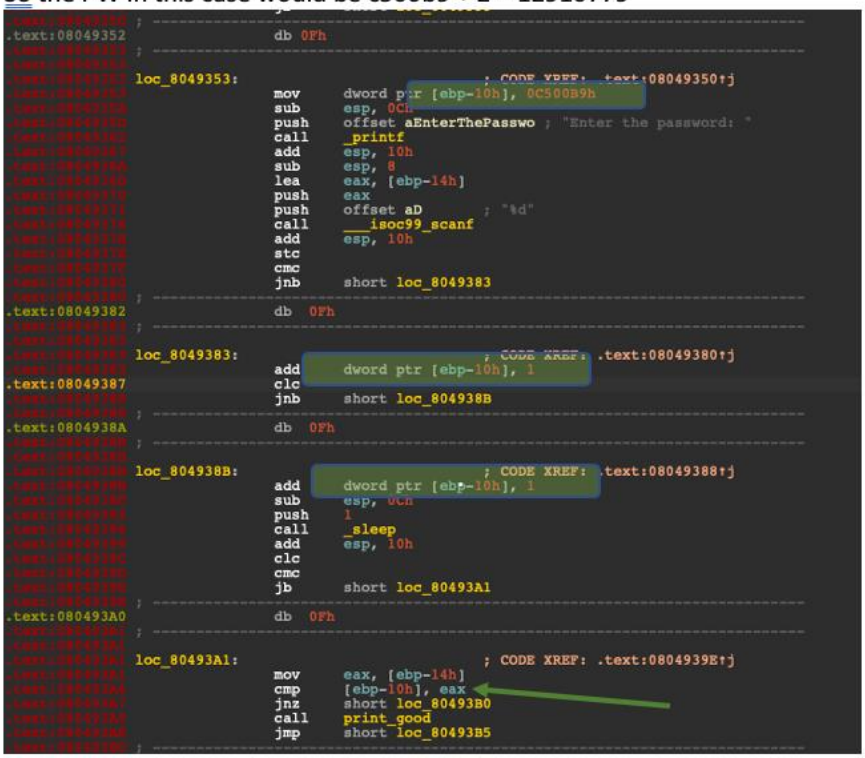
- Right-click -> undefine, the function at the end of main (in my case loc_8049344)
- This changes the function's name, right click the newly named function and change to code
- Your screen should look like the screenshot to the right
- You can see a hex value being mov'd into [ebp-10h], convert that hex -> decimal.



- Open binary in IDA, go to main
- Whenever you see a jump to location + 1 (or any number), you need to convert the target functions (highlighted green) to undefined, then convert to data
- Once you do it to these two functions, more of them will appear, just follow the same steps



- Eventually you should get something that looks like this..
- The program prints good whenever ebp-10h matches eax (at the bottom)
- At the top some hex value is loaded into ebp-10h, along the way it gets incremented
- So the PW in this case would be c500b9 + 2 = 12910779



- In IDA pro observe the main function
- A hex value is being stored in ebp-14 which later gets bitshifted (least significant bit) by 4, then is compared, jumping to print_good if successfully matched
- Goto a bitshift calculator website, enter your hex value in my case 0xA90AAC, shift left by 4, the decimal produced is your PW

```
; int __cdecl main(int argc, const char **argv, const char **envp)
                public main
main:                                   ; DATA XREF: _start+2A↑o
                lea     ecx, [esp+4]
                and     esp, 0FFFFFFF0h
                push    dword ptr [ecx-4]
                push    ebp
                mov     ebp, esp
                push    ecx
                sub     esp, 24h
                mov     eax, ecx
                mov     eax, [eax+4]
                mov     [ebp-1Ch], eax
                mov     eax, large gs:14h
                mov     [ebp-0Ch], eax
                xor     eax, eax
                call    print_msg
                sub     esp, 0Ch
                push    offset aEnterThePasswo ; "Enter the password: "
                call    _printf
                add     esp, 10h
                sub     esp, 8
                lea     eax, [ebp-18h]
                push    eax
                push    offset aD       ; "%d"
                call    ___isoc99_scanf
                add     esp, 10h
                mov     dword ptr [ebp-14h], 0A90AACh
                mov     eax, 2

loc_804936F:                            ; CODE XREF: .text:loc_804936F↑j
                jmp     short near ptr loc_804936F+1
; ---------------------------------------------------------------------------
                sar     bh, 0C0h
                mov     [ebp-10h], eax
                mov     eax, [ebp-10h]
                mov     ecx, eax
                shl     dword ptr [ebp-14h], cl
                mov     eax, [ebp-18h]
                cmp     [ebp-14h], eax
                jnz     short loc_804938E
                call    print_good
                jmp     short loc_8049393
```

Bit Shift Calculator

Perform bit shift operations with decimal, hexadecimal, binary and octal numbers

Number Base

Hexadecimal (16)

Number

A90AAC

Shift Direction

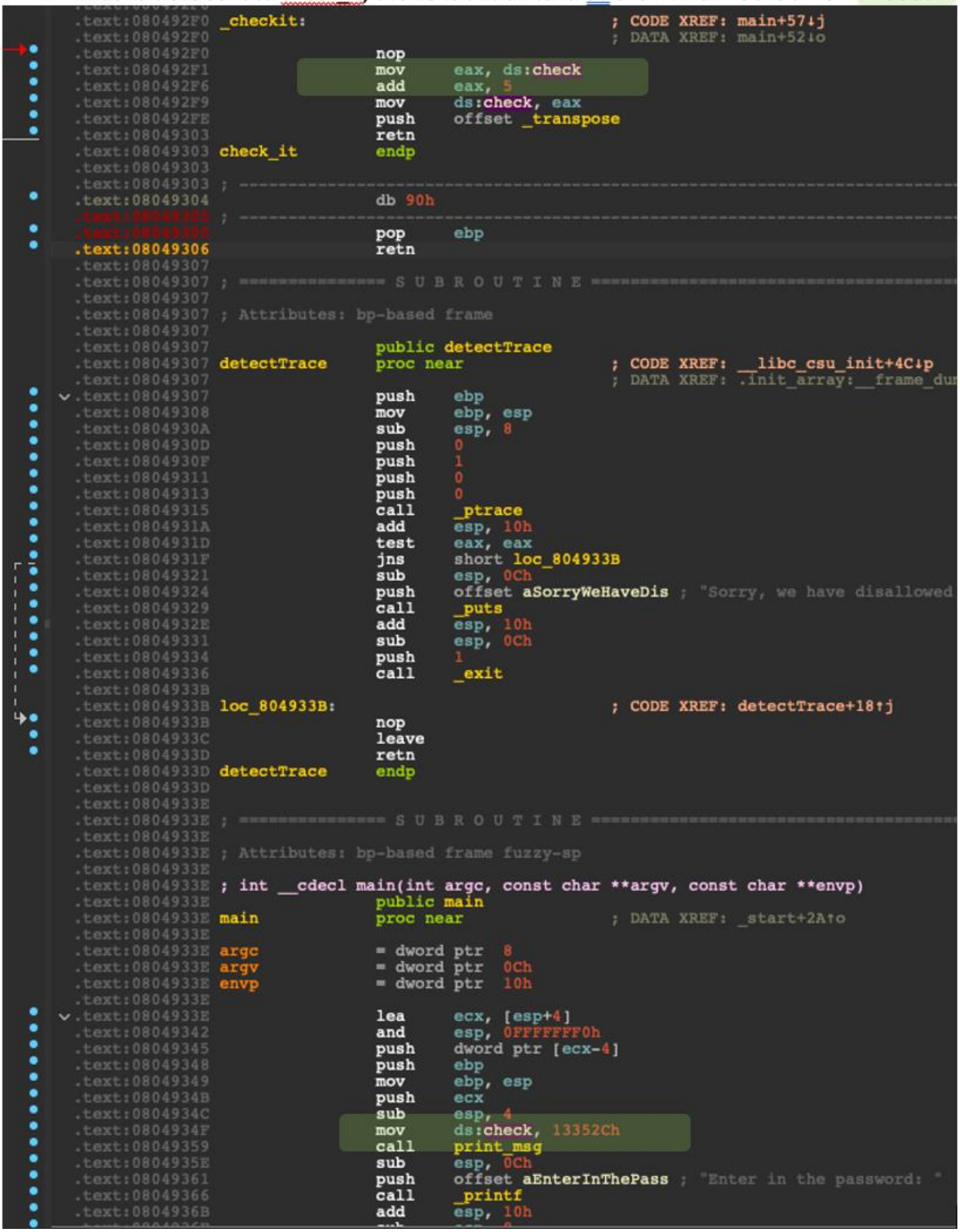
<< Left

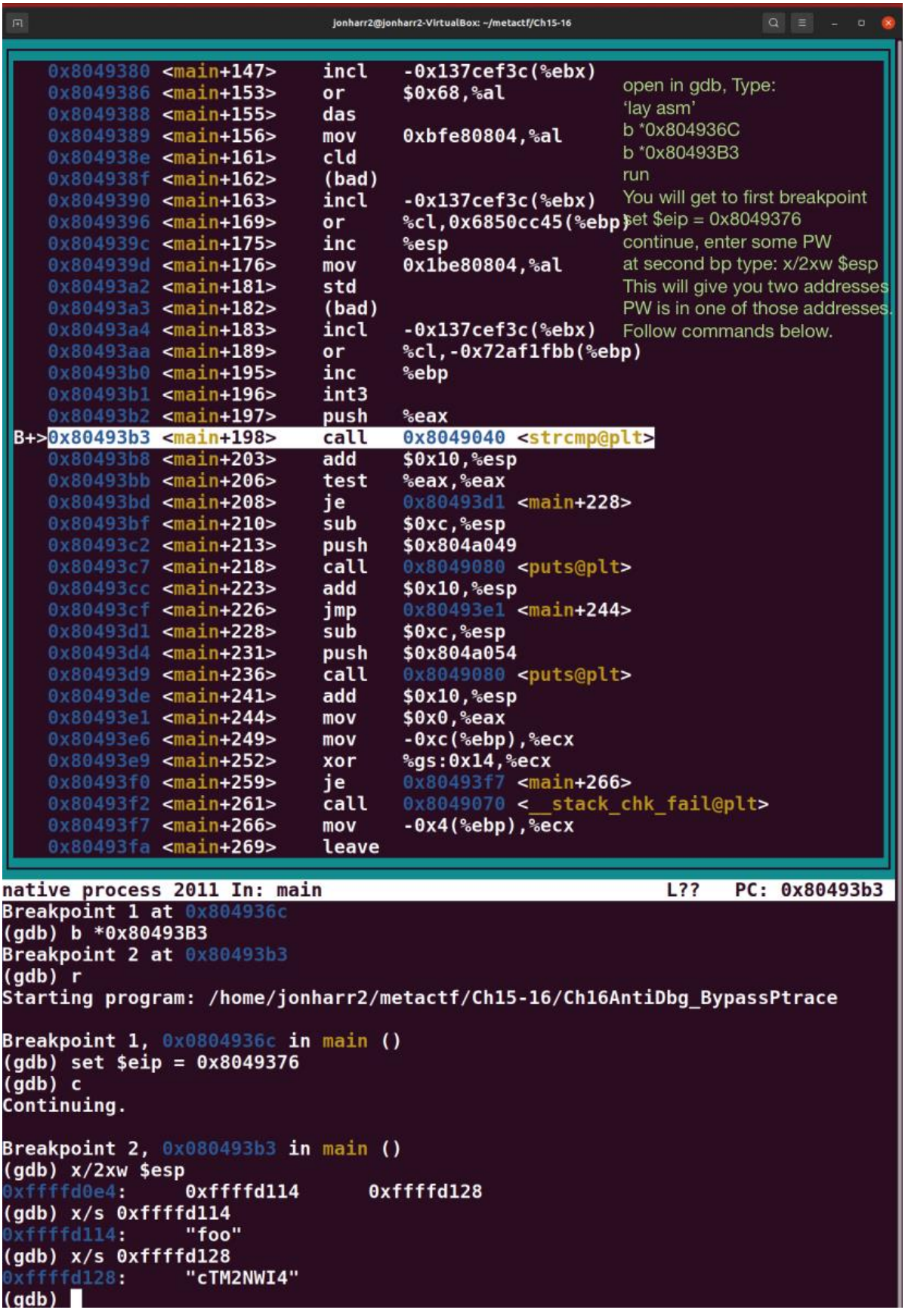
Bits to Shift

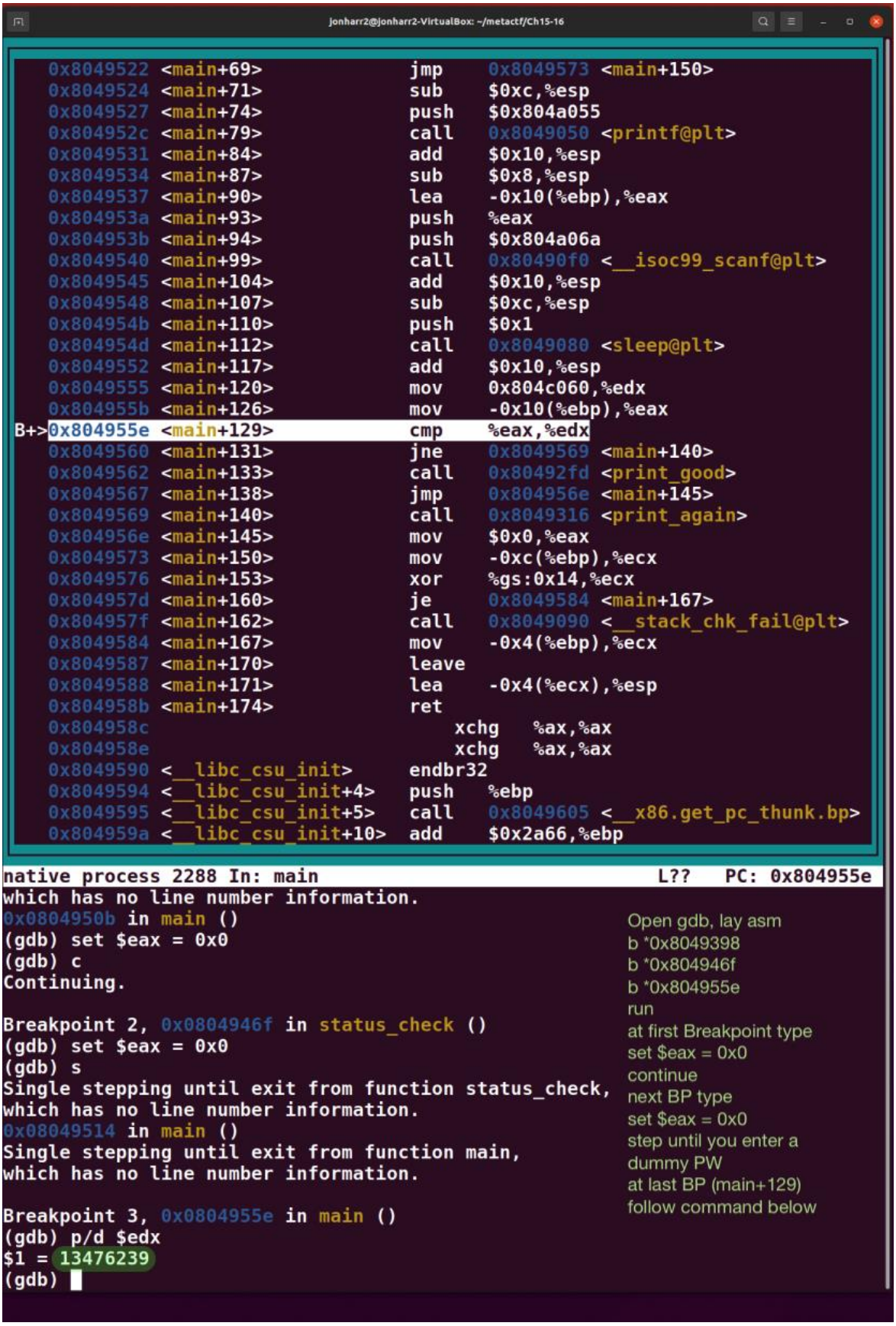
4

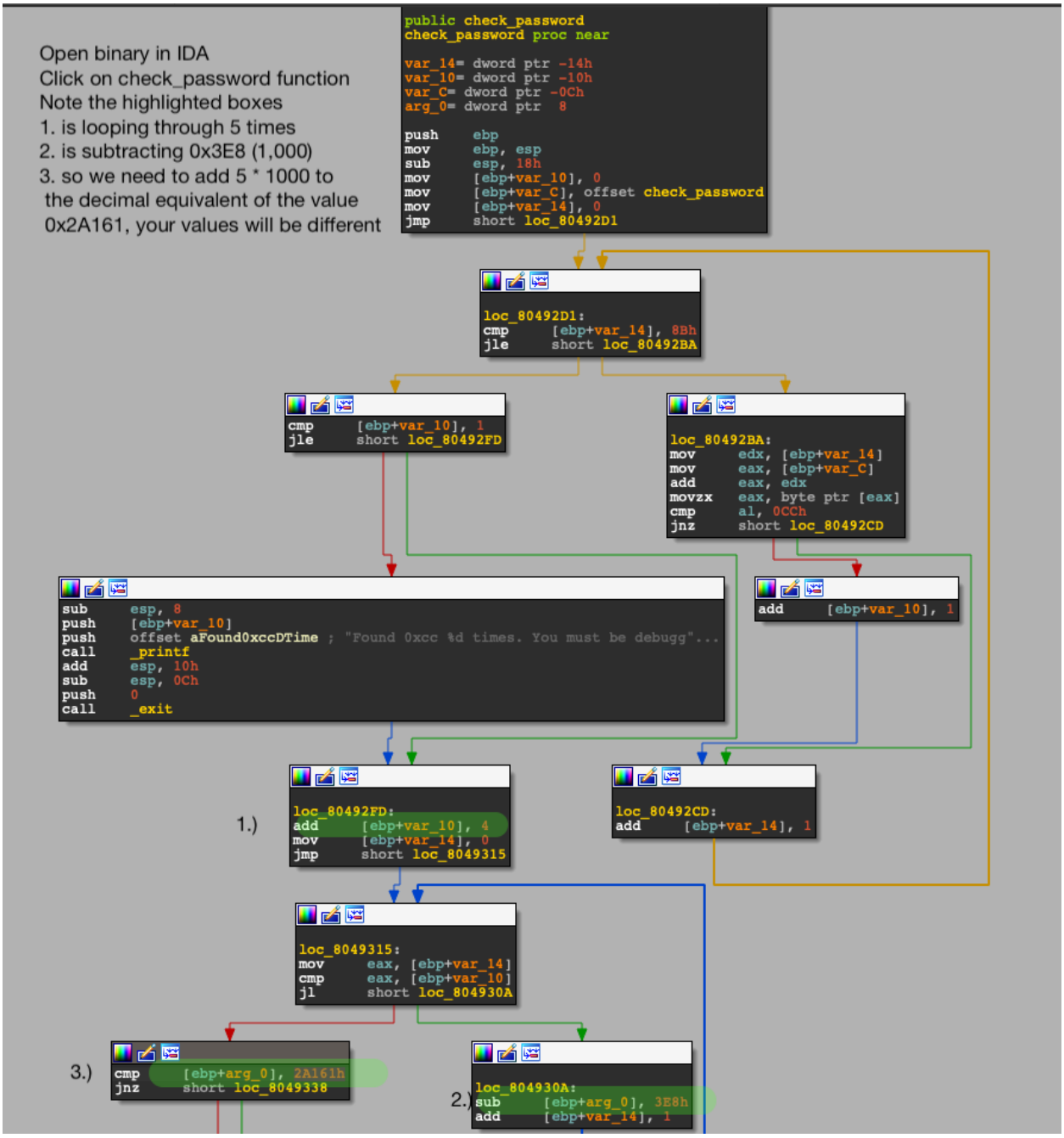
Decimal	177253056
Binary	10101001000010101011000000
Hexadecimal	a90aac0
Octal	1244125300

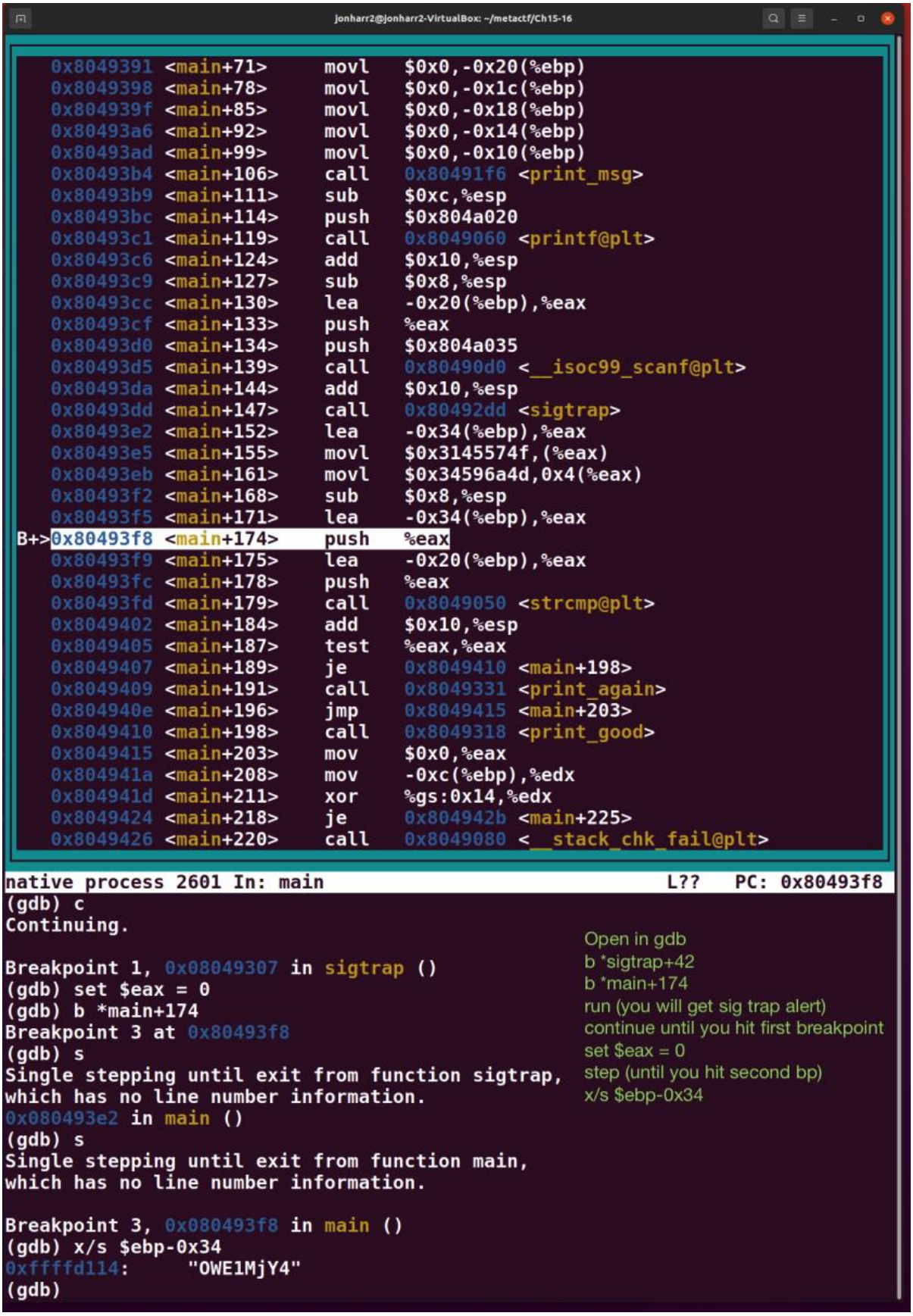
- Open Binary in IDA Pro
- Some hex value is being moved into the function check (0x11352C) *bottom box
- That value is sent to check_it, then 5 is added to it. So the PW is 1258796 + 5 = 1258801




jonharr2@jonharr2-VirtualBox: ~/metactf/Ch15-16
0x8049380 <main+147> incl -0x137cef3c(%ebx)
0x8049386 <main+153> or $0x68,%al
0x8049388 <main+155> das
0x8049389 <main+156> mov 0xbfe80804,%al
0x804938e <main+161> cld
0x804938f <main+162> (bad)
0x8049390 <main+163> incl -0x137cef3c(%ebx)
0x8049396 <main+169> or %cl,0x6850cc45(%ebp)
0x804939c <main+175> inc %esp
0x804939d <main+176> mov 0x1be80804,%al
0x80493a2 <main+181> std
0x80493a3 <main+182> (bad)
0x80493a4 <main+183> incl -0x137cef3c(%ebx)
0x80493aa <main+189> or %cl,-0x72af1fbb(%ebp)
0x80493b0 <main+195> inc %ebp
0x80493b1 <main+196> int3
0x80493b2 <main+197> push %eax
B+>0x80493b3 <main+198> call 0x8049040 <strcmp@plt>
0x80493b8 <main+203> add $0x10,%esp
0x80493bb <main+206> test %eax,%eax
0x80493bd <main+208> je 0x80493d1 <main+228>
0x80493bf <main+210> sub $0xc,%esp
0x80493c2 <main+213> push $0x804a049
0x80493c7 <main+218> call 0x8049080 <puts@plt>
0x80493cc <main+223> add $0x10,%esp
0x80493cf <main+226> jmp 0x80493e1 <main+244>
0x80493d1 <main+228> sub $0xc,%esp
0x80493d4 <main+231> push $0x804a054
0x80493d9 <main+236> call 0x8049080 <puts@plt>
0x80493de <main+241> add $0x10,%esp
0x80493e1 <main+244> mov $0x0,%eax
0x80493e6 <main+249> mov -0xc(%ebp),%ecx
0x80493e9 <main+252> xor %gs:0x14,%ecx
0x80493f0 <main+259> je 0x80493f7 <main+266>
0x80493f2 <main+261> call 0x8049070 <__stack_chk_fail@plt>
0x80493f7 <main+266> mov -0x4(%ebp),%ecx
0x80493fa <main+269> leave
open in gdb, Type:
'lay asm'
b *0x804936C
b *0x80493B3
run
You will get to first breakpoint
set $eip = 0x8049376
continue, enter some PW
at second bp type: x/2xw $esp
This will give you two addresses
PW is in one of those addresses.
Follow commands below.
native process 2011 In: main L?? PC: 0x80493b3
Breakpoint 1 at 0x804936c
(gdb) b *0x80493B3
Breakpoint 2 at 0x80493b3
(gdb) r
Starting program: /home/jonharr2/metactf/Ch15-16/Ch16AntiDbg_BypassPtrace
Breakpoint 1, 0x0804936c in main ()
(gdb) set $eip = 0x8049376
(gdb) c
Continuing.
Breakpoint 2, 0x080493b3 in main ()
(gdb) x/2xw $esp
0xffffd0e4: 0xffffd114 0xffffd128
(gdb) x/s 0xffffd114
0xffffd114: "foo"
(gdb) x/s 0xffffd128
0xffffd128: "cTM2NWI4"
(gdb)

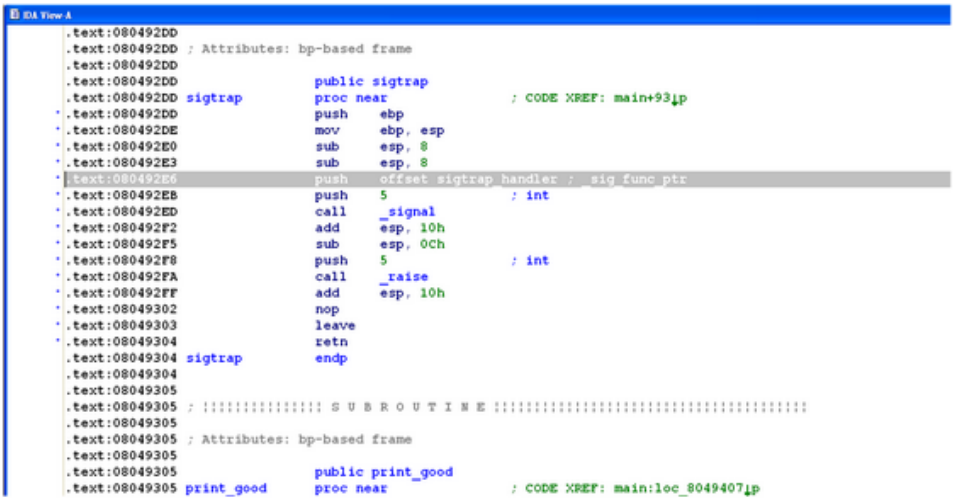

jonharr2@jonharr2-VirtualBox: ~/metactf/Ch15-16
0x8049522 <main+69> jmp 0x8049573 <main+150>
0x8049524 <main+71> sub $0xc,%esp
0x8049527 <main+74> push $0x804a055
0x804952c <main+79> call 0x8049050 <printf@plt>
0x8049531 <main+84> add $0x10,%esp
0x8049534 <main+87> sub $0x8,%esp
0x8049537 <main+90> lea -0x10(%ebp),%eax
0x804953a <main+93> push %eax
0x804953b <main+94> push $0x804a06a
0x8049540 <main+99> call 0x80490f0 <__isoc99_scanf@plt>
0x8049545 <main+104> add $0x10,%esp
0x8049548 <main+107> sub $0xc,%esp
0x804954b <main+110> push $0x1
0x804954d <main+112> call 0x8049080 <sleep@plt>
0x8049552 <main+117> add $0x10,%esp
0x8049555 <main+120> mov 0x804c060,%edx
0x804955b <main+126> mov -0x10(%ebp),%eax
B+>0x804955e <main+129> cmp %eax,%edx
0x8049560 <main+131> jne 0x8049569 <main+140>
0x8049562 <main+133> call 0x80492fd <print_good>
0x8049567 <main+138> jmp 0x804956e <main+145>
0x8049569 <main+140> call 0x8049316 <print_again>
0x804956e <main+145> mov $0x0,%eax
0x8049573 <main+150> mov -0xc(%ebp),%ecx
0x8049576 <main+153> xor %gs:0x14,%ecx
0x804957d <main+160> je 0x8049584 <main+167>
0x804957f <main+162> call 0x8049090 <__stack_chk_fail@plt>
0x8049584 <main+167> mov -0x4(%ebp),%ecx
0x8049587 <main+170> leave
0x8049588 <main+171> lea -0x4(%ecx),%esp
0x804958b <main+174> ret
0x804958c xchg %ax,%ax
0x804958e xchg %ax,%ax
0x8049590 <__libc_csu_init> endbr32
0x8049594 <__libc_csu_init+4> push %ebp
0x8049595 <__libc_csu_init+5> call 0x8049605 <__x86.get_pc_thunk.bp>
0x804959a <__libc_csu_init+10> add $0x2a66,%ebp
native process 2288 In: main L?? PC: 0x804955e
which has no line number information.
0x0804950b in main ()
(gdb) set $eax = 0x0
(gdb) c
Continuing.
Breakpoint 2, 0x0804946f in status_check ()
(gdb) set $eax = 0x0
(gdb) s
Single stepping until exit from function status_check,
which has no line number information.
0x08049514 in main ()
Single stepping until exit from function main,
which has no line number information.
Breakpoint 3, 0x0804955e in main ()
(gdb) p/d $edx
$1 = 13476239
(gdb)
Open gdb, lay asm
b *0x8049398
b *0x804946f
b *0x804955e
run
at first Breakpoint type
set $eax = 0x0
continue
next BP type
set $eax = 0x0
step until you enter a
dummy PW
at last BP (main+129)
follow command below


Open binary in IDA
Click on check_password function
Note the highlighted boxes
1. is looping through 5 times
2. is subtracting 0x3E8 (1,000)
3. so we need to add 5 * 1000 to
the decimal equivalent of the value
0x2A161, your values will be different
public check_password
check_password proc near
var_14= dword ptr -14h
var_10= dword ptr -10h
var_C= dword ptr -0Ch
arg_0= dword ptr 8
push ebp
mov ebp, esp
sub esp, 18h
mov [ebp+var_10], 0
mov [ebp+var_C], offset check_password
mov [ebp+var_14], 0
jmp short loc_80492D1
loc_80492D1:
cmp [ebp+var_14], 8Bh
jle short loc_80492BA
cmp [ebp+var_10], 1
jle short loc_80492FD
loc_80492BA:
mov edx, [ebp+var_14]
mov eax, [ebp+var_C]
add eax, edx
movzx eax, byte ptr [eax]
cmp al, 0CCh
jnz short loc_80492CD
sub esp, 8
push [ebp+var_10]
push offset aFound0xccDTime ; "Found 0xcc %d times. You must be debugg"...
call _printf
add esp, 10h
sub esp, 0Ch
push 0
call _exit
add [ebp+var_10], 1
1.)
loc_80492FD:
add [ebp+var_10], 4
mov [ebp+var_14], 0
jmp short loc_8049315
loc_80492CD:
add [ebp+var_14], 1
loc_8049315:
mov eax, [ebp+var_14]
cmp eax, [ebp+var_10]
jl short loc_804930A
3.)
cmp [ebp+arg_0], 2A161h
jnz short loc_8049338
2.)
loc_804930A:
sub [ebp+arg_0], 3E8h
add [ebp+var_14], 1

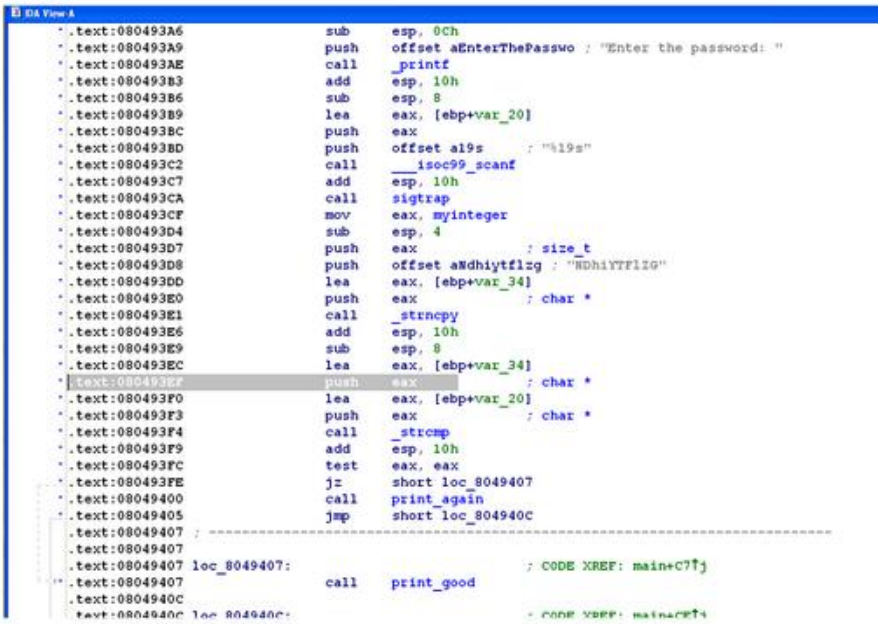

jonharr2@jonharr2-VirtualBox: ~/metactf/Ch15-16
0x8049391 <main+71> movl $0x0,-0x20(%ebp)
0x8049398 <main+78> movl $0x0,-0x1c(%ebp)
0x804939f <main+85> movl $0x0,-0x18(%ebp)
0x80493a6 <main+92> movl $0x0,-0x14(%ebp)
0x80493ad <main+99> movl $0x0,-0x10(%ebp)
0x80493b4 <main+106> call 0x80491f6 <print_msg>
0x80493b9 <main+111> sub $0xc,%esp
0x80493bc <main+114> push $0x804a020
0x80493c1 <main+119> call 0x8049060 <printf@plt>
0x80493c6 <main+124> add $0x10,%esp
0x80493c9 <main+127> sub $0x8,%esp
0x80493cc <main+130> lea -0x20(%ebp),%eax
0x80493cf <main+133> push %eax
0x80493d0 <main+134> push $0x804a035
0x80493d5 <main+139> call 0x80490d0 <__isoc99_scanf@plt>
0x80493da <main+144> add $0x10,%esp
0x80493dd <main+147> call 0x80492dd <sigtrap>
0x80493e2 <main+152> lea -0x34(%ebp),%eax
0x80493e5 <main+155> movl $0x3145574f,(%eax)
0x80493eb <main+161> movl $0x34596a4d,0x4(%eax)
0x80493f2 <main+168> sub $0x8,%esp
0x80493f5 <main+171> lea -0x34(%ebp),%eax
B+>0x80493f8 <main+174> push %eax
0x80493f9 <main+175> lea -0x20(%ebp),%eax
0x80493fc <main+178> push %eax
0x80493fd <main+179> call 0x8049050 <strcmp@plt>
0x8049402 <main+184> add $0x10,%esp
0x8049405 <main+187> test %eax,%eax
0x8049407 <main+189> je 0x8049410 <main+198>
0x8049409 <main+191> call 0x8049331 <print_again>
0x804940e <main+196> jmp 0x8049415 <main+203>
0x8049410 <main+198> call 0x8049318 <print_good>
0x8049415 <main+203> mov $0x0,%eax
0x804941a <main+208> mov -0xc(%ebp),%edx
0x804941d <main+211> xor %gs:0x14,%edx
0x8049424 <main+218> je 0x804942b <main+225>
0x8049426 <main+220> call 0x8049080 <__stack_chk_fail@plt>
native process 2601 In: main L?? PC: 0x80493f8
(gdb) c
Continuing.
Breakpoint 1, 0x08049307 in sigtrap ()
(gdb) set $eax = 0
(gdb) b *main+174
Breakpoint 3 at 0x80493f8
(gdb) s
Single stepping until exit from function sigtrap,
which has no line number information.
0x080493e2 in main ()
(gdb) s
Single stepping until exit from function main,
which has no line number information.
Breakpoint 3, 0x080493f8 in main ()
(gdb) x/s $ebp-0x34
0xffffd114: "OWE1MjY4"
(gdb)
Open in gdb
b *sigtrap+42
b *main+174
run (you will get sig trap alert)
continue until you hit first breakpoint
set $eax = 0
step (until you hit second bp)
x/s $ebp-0x34

5. **Ch16AntiDbg_SigtrapEntangle**
    a. Open in IDA

    b. Go to subroutine “sigtrap” and find the highlighted line below. Make note of line number.
        i. Also make note of the first line number address of the sigtrap subroutine as you will need to enter this as a breakpoint so the program will break at the first line in the subroutine.



    c. Go back out into main and find the highlighted line below and make note of the line number.



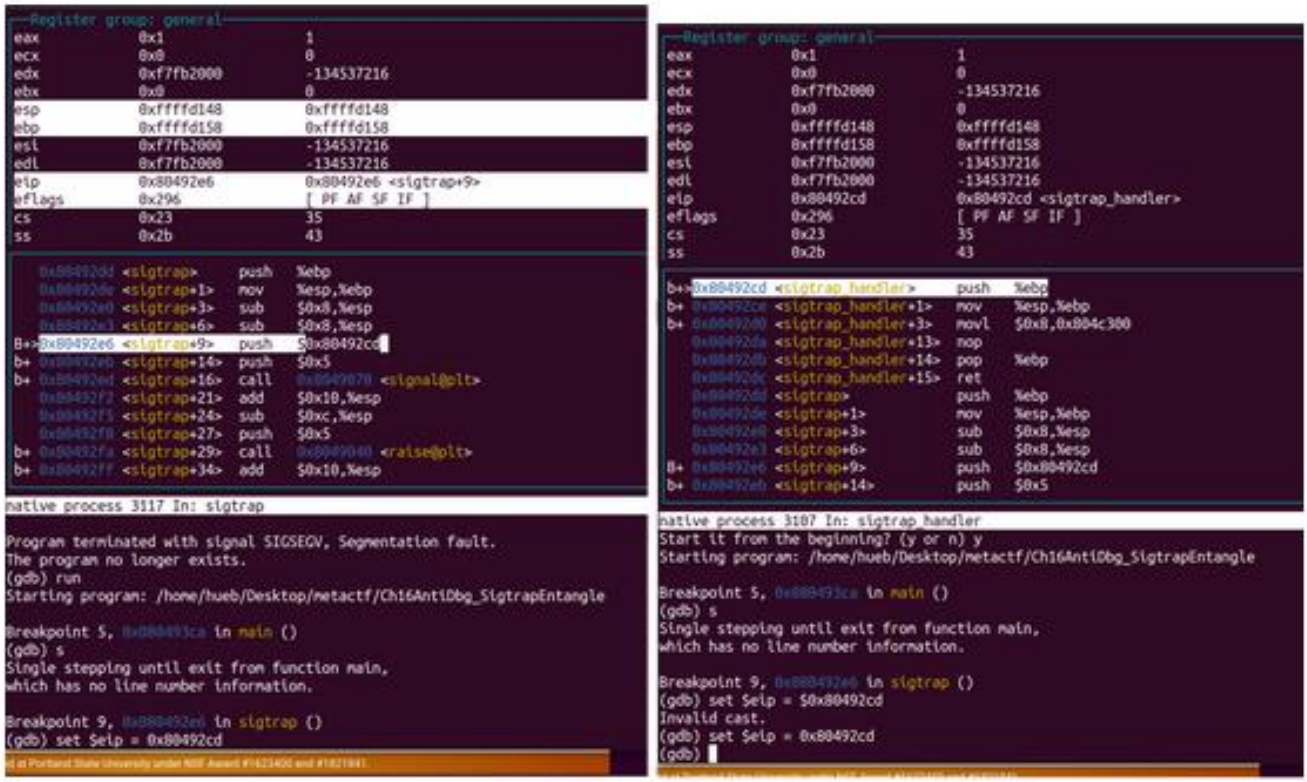
    d. Open in GDB

    e. Set 2 breakpoints for the line numbers found in step “b” and another breakpoint for the line number found in step “c”.

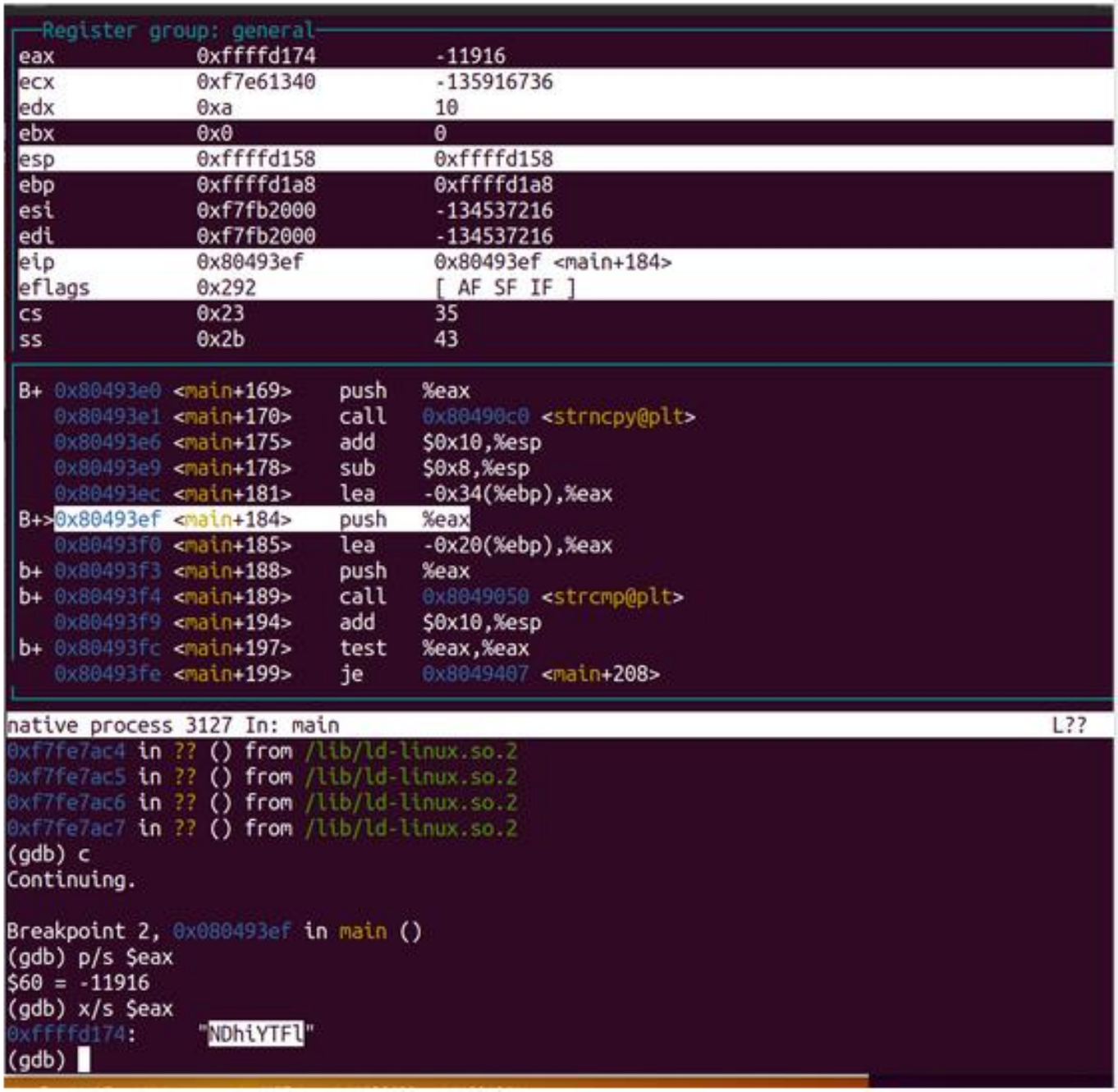
    f. Run in GDB

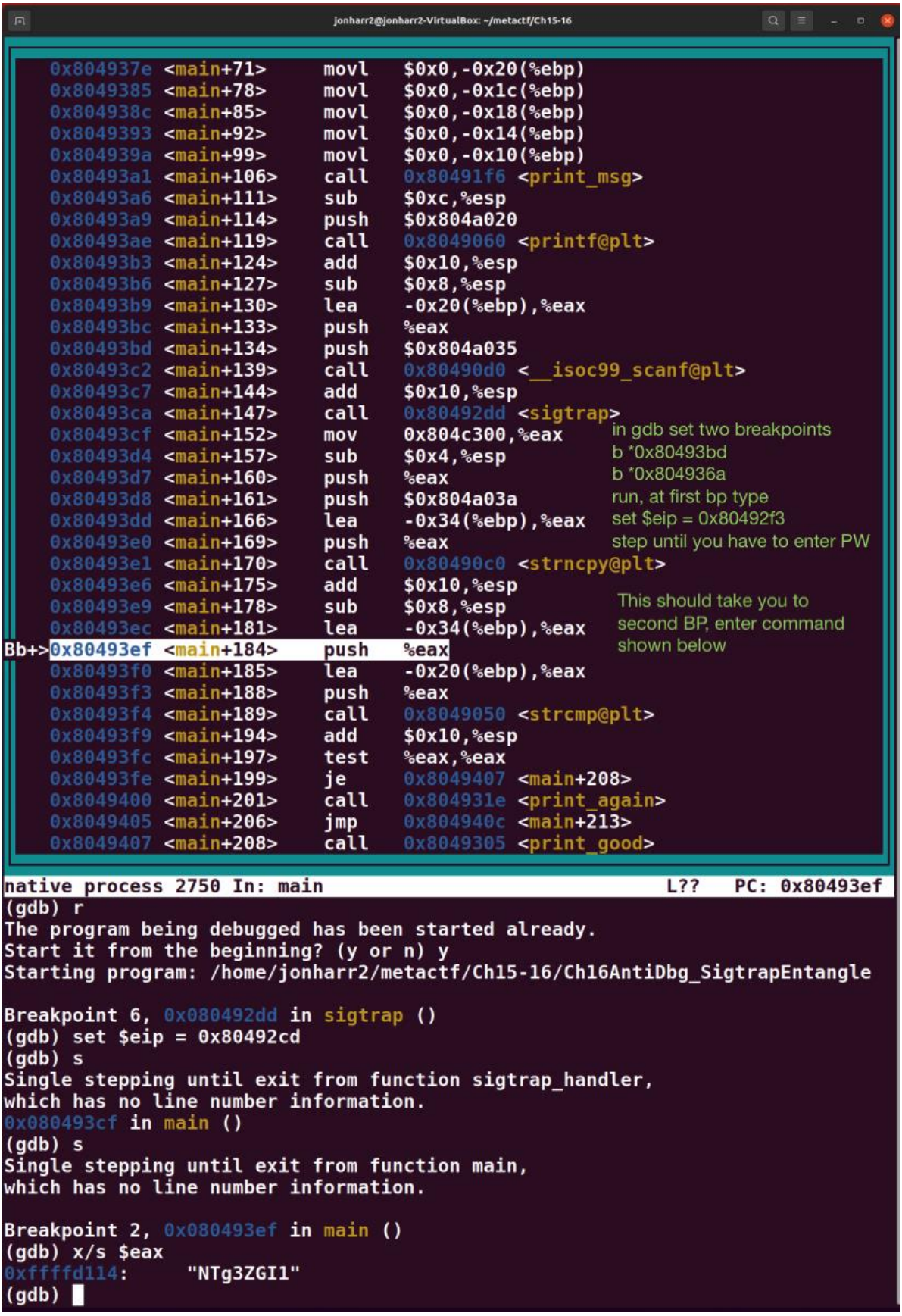
    g. Enter bogus password “foo”

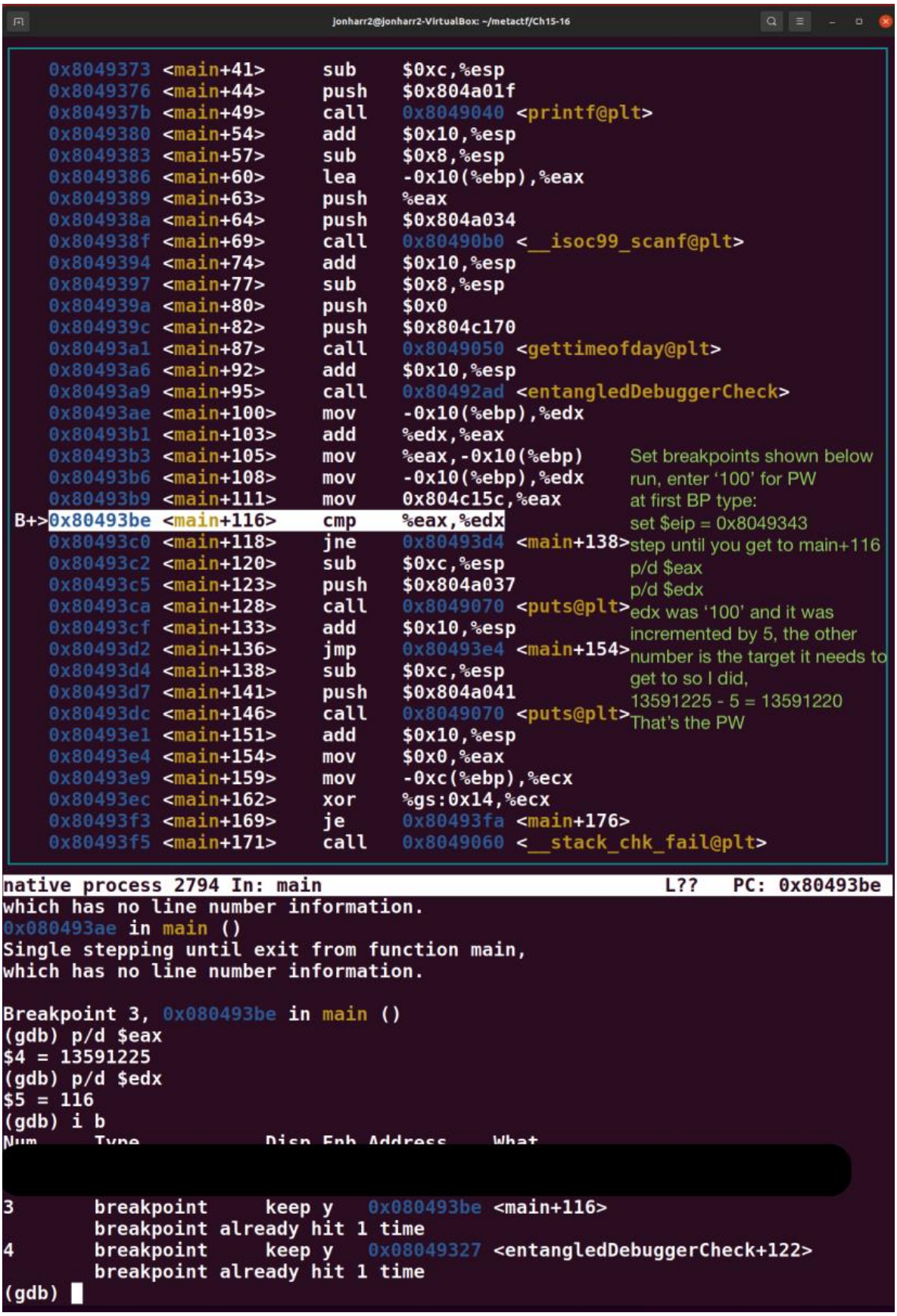
h. Once you hit the first breakpoint found in step “b”, where you first enter the “sigtrap” function, find the push instruction mentioned in step b, which should also have a breakpoint on it.
    i. Set the instruction pointer to the hex address being pushed on that line in GDB. This should bring you to the sigtrap_handler section. Now just keep stepping or continue until you hit the next breakpoint.

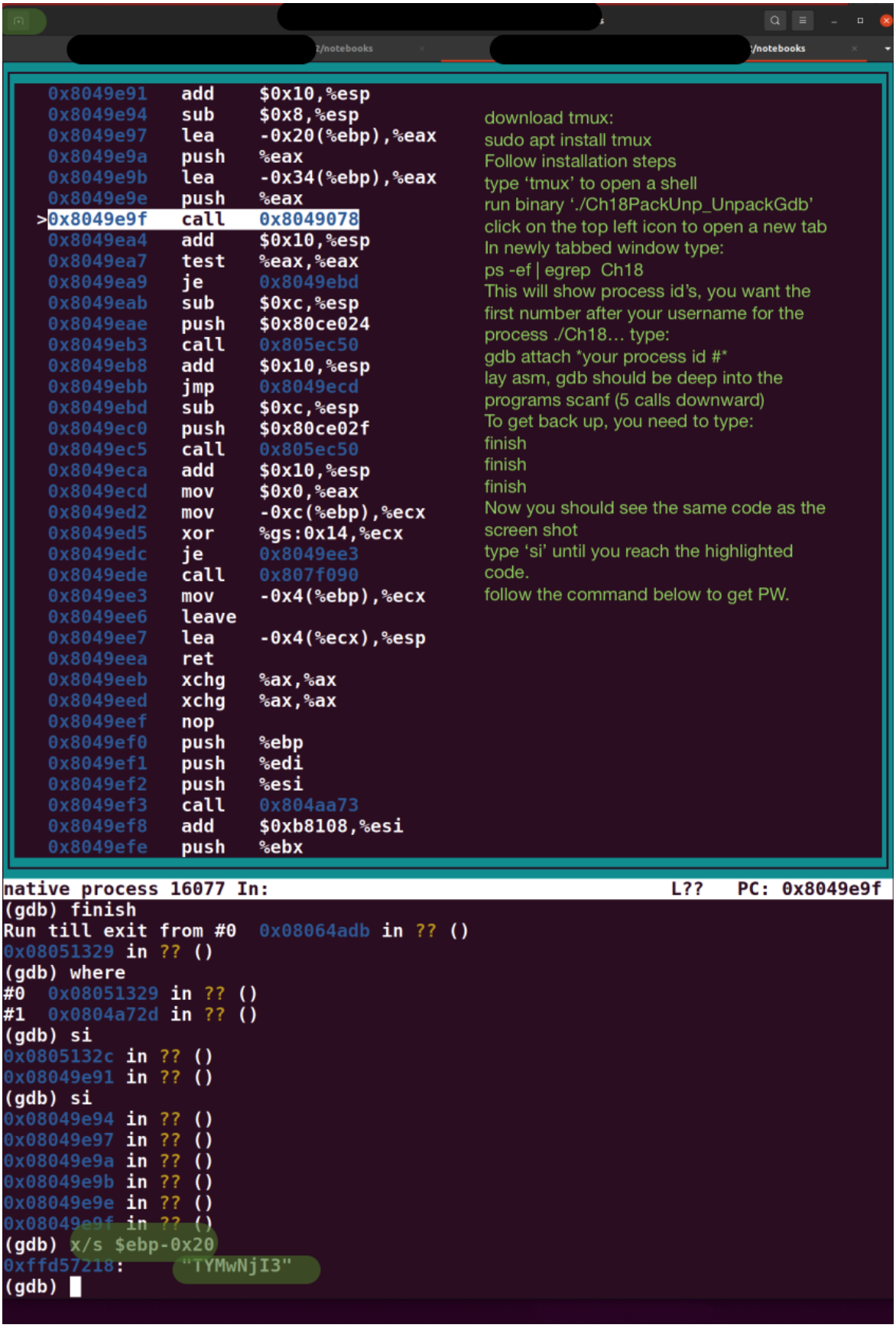


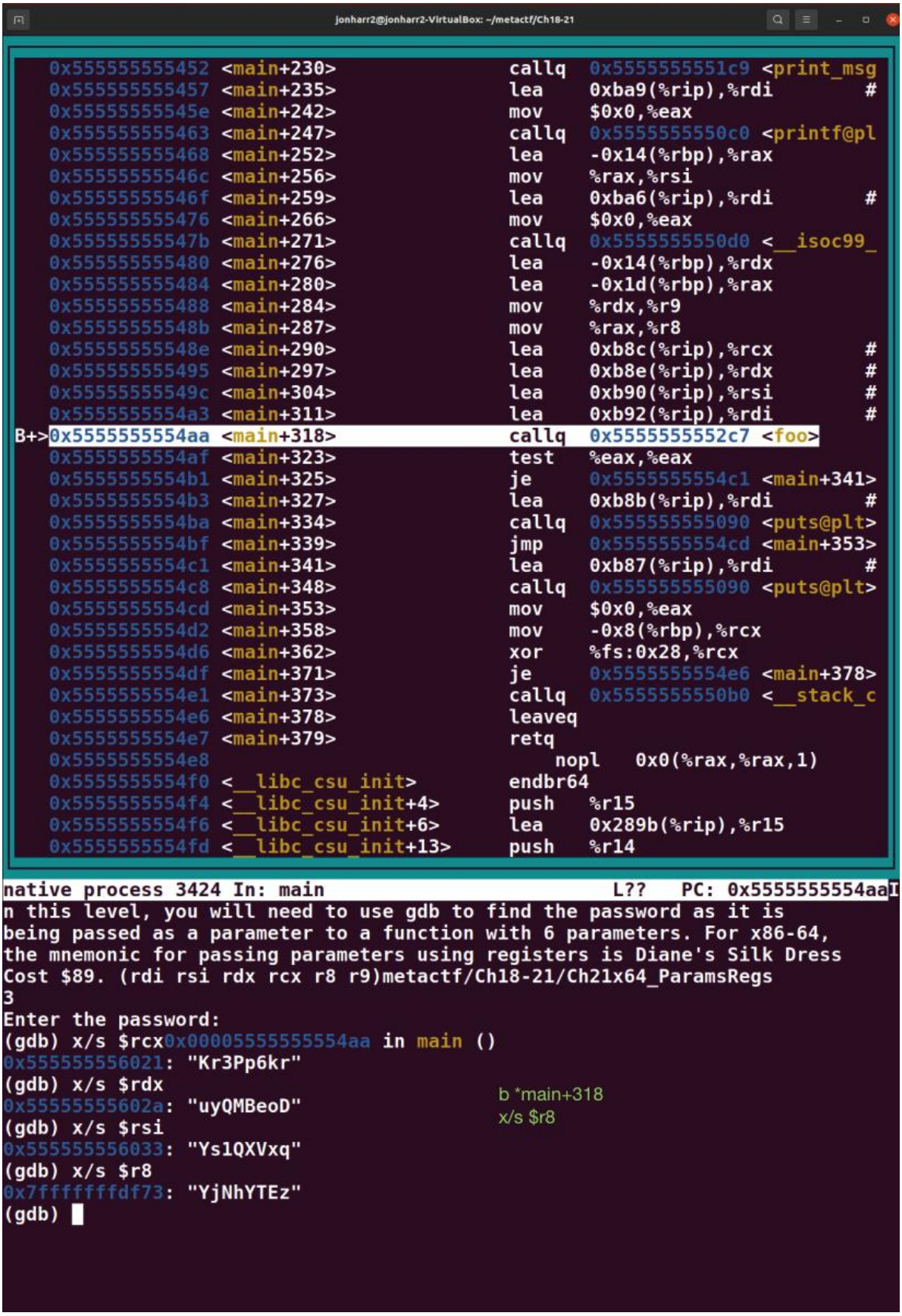
i. Once you hit the final breakpoint where eax is being pushed just use the command “x/s $eax” to print out the password.




jonharr2@jonharr2-VirtualBox: ~/metactf/Ch15-16
0x804937e <main+71> movl $0x0,-0x20(%ebp)
0x8049385 <main+78> movl $0x0,-0x1c(%ebp)
0x804938c <main+85> movl $0x0,-0x18(%ebp)
0x8049393 <main+92> movl $0x0,-0x14(%ebp)
0x804939a <main+99> movl $0x0,-0x10(%ebp)
0x80493a1 <main+106> call 0x80491f6 <print_msg>
0x80493a6 <main+111> sub $0xc,%esp
0x80493a9 <main+114> push $0x804a020
0x80493ae <main+119> call 0x8049060 <printf@plt>
0x80493b3 <main+124> add $0x10,%esp
0x80493b6 <main+127> sub $0x8,%esp
0x80493b9 <main+130> lea -0x20(%ebp),%eax
0x80493bc <main+133> push %eax
0x80493bd <main+134> push $0x804a035
0x80493c2 <main+139> call 0x80490d0 <__isoc99_scanf@plt>
0x80493c7 <main+144> add $0x10,%esp
0x80493ca <main+147> call 0x80492dd <sigtrap>
0x80493cf <main+152> mov 0x804c300,%eax
0x80493d4 <main+157> sub $0x4,%esp
0x80493d7 <main+160> push %eax
0x80493d8 <main+161> push $0x804a03a
0x80493dd <main+166> lea -0x34(%ebp),%eax
0x80493e0 <main+169> push %eax
0x80493e1 <main+170> call 0x80490c0 <strncpy@plt>
0x80493e6 <main+175> add $0x10,%esp
0x80493e9 <main+178> sub $0x8,%esp
0x80493ec <main+181> lea -0x34(%ebp),%eax
Bb+> 0x80493ef <main+184> push %eax
0x80493f0 <main+185> lea -0x20(%ebp),%eax
0x80493f3 <main+188> push %eax
0x80493f4 <main+189> call 0x8049050 <strcmp@plt>
0x80493f9 <main+194> add $0x10,%esp
0x80493fc <main+197> test %eax,%eax
0x80493fe <main+199> je 0x8049407 <main+208>
0x8049400 <main+201> call 0x804931e <print_again>
0x8049405 <main+206> jmp 0x804940c <main+213>
0x8049407 <main+208> call 0x8049305 <print_good>
in gdb set two breakpoints
b *0x80493bd
b *0x804936a
run, at first bp type
set $eip = 0x80492f3
step until you have to enter PW
This should take you to
second BP, enter command
shown below
native process 2750 In: main L?? PC: 0x80493ef
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/jonharr2/metactf/Ch15-16/Ch16AntiDbg_SigtrapEntangle
Breakpoint 6, 0x080492dd in sigtrap ()
(gdb) set $eip = 0x80492cd
(gdb) s
Single stepping until exit from function sigtrap_handler,
which has no line number information.
0x080493cf in main ()
(gdb) s
Single stepping until exit from function main,
which has no line number information.
Breakpoint 2, 0x080493ef in main ()
(gdb) x/s $eax
0xffffd114: "NTg3ZGI1"
(gdb)


jonharr2@jonharr2-VirtualBox: ~/metactf/Ch15-16
0x8049373 <main+41> sub $0xc,%esp
0x8049376 <main+44> push $0x804a01f
0x804937b <main+49> call 0x8049040 <printf@plt>
0x8049380 <main+54> add $0x10,%esp
0x8049383 <main+57> sub $0x8,%esp
0x8049386 <main+60> lea -0x10(%ebp),%eax
0x8049389 <main+63> push %eax
0x804938a <main+64> push $0x804a034
0x804938f <main+69> call 0x80490b0 <__isoc99_scanf@plt>
0x8049394 <main+74> add $0x10,%esp
0x8049397 <main+77> sub $0x8,%esp
0x804939a <main+80> push $0x0
0x804939c <main+82> push $0x804c170
0x80493a1 <main+87> call 0x8049050 <gettimeofday@plt>
0x80493a6 <main+92> add $0x10,%esp
0x80493a9 <main+95> call 0x80492ad <entangledDebuggerCheck>
0x80493ae <main+100> mov -0x10(%ebp),%edx
0x80493b1 <main+103> add %edx,%eax
0x80493b3 <main+105> mov %eax,-0x10(%ebp)
0x80493b6 <main+108> mov -0x10(%ebp),%edx
0x80493b9 <main+111> mov 0x804c15c,%eax
B+>0x80493be <main+116> cmp %eax,%edx
0x80493c0 <main+118> jne 0x80493d4 <main+138>
0x80493c2 <main+120> sub $0xc,%esp
0x80493c5 <main+123> push $0x804a037
0x80493ca <main+128> call 0x8049070 <puts@plt>
0x80493cf <main+133> add $0x10,%esp
0x80493d2 <main+136> jmp 0x80493e4 <main+154>
0x80493d4 <main+138> sub $0xc,%esp
0x80493d7 <main+141> push $0x804a041
0x80493dc <main+146> call 0x8049070 <puts@plt>
0x80493e1 <main+151> add $0x10,%esp
0x80493e4 <main+154> mov $0x0,%eax
0x80493e9 <main+159> mov -0xc(%ebp),%ecx
0x80493ec <main+162> xor %gs:0x14,%ecx
0x80493f3 <main+169> je 0x80493fa <main+176>
0x80493f5 <main+171> call 0x8049060 <__stack_chk_fail@plt>
Set breakpoints shown below
run, enter '100' for PW
at first BP type:
set $eip = 0x8049343
step until you get to main+116
p/d $eax
p/d $edx
edx was '100' and it was
incremented by 5, the other
number is the target it needs to
get to so I did,
13591225 - 5 = 13591220
That's the PW
native process 2794 In: main L?? PC: 0x80493be
which has no line number information.
0x080493ae in main ()
Single stepping until exit from function main,
which has no line number information.
Breakpoint 3, 0x080493be in main ()
(gdb) p/d $eax
$4 = 13591225
(gdb) p/d $edx
$5 = 116
(gdb) i b
Num Type Disp Enb Address What
3 breakpoint keep y 0x080493be <main+116>
breakpoint already hit 1 time
4 breakpoint keep y 0x08049327 <entangledDebuggerCheck+122>
breakpoint already hit 1 time
(gdb)


0x8049e91 add $0x10,%esp
0x8049e94 sub $0x8,%esp
0x8049e97 lea -0x20(%ebp),%eax
0x8049e9a push %eax
0x8049e9b lea -0x34(%ebp),%eax
0x8049e9e push %eax
>0x8049e9f call 0x8049078
0x8049ea4 add $0x10,%esp
0x8049ea7 test %eax,%eax
0x8049ea9 je 0x8049ebd
0x8049eab sub $0xc,%esp
0x8049eae push $0x80ce024
0x8049eb3 call 0x805ec50
0x8049eb8 add $0x10,%esp
0x8049ebb jmp 0x8049ecd
0x8049ebd sub $0xc,%esp
0x8049ec0 push $0x80ce02f
0x8049ec5 call 0x805ec50
0x8049eca add $0x10,%esp
0x8049ecd mov $0x0,%eax
0x8049ed2 mov -0xc(%ebp),%ecx
0x8049ed5 xor %gs:0x14,%ecx
0x8049edc je 0x8049ee3
0x8049ede call 0x807f090
0x8049ee3 mov -0x4(%ebp),%ecx
0x8049ee6 leave
0x8049ee7 lea -0x4(%ecx),%esp
0x8049eea ret
0x8049eeb xchg %ax,%ax
0x8049eed xchg %ax,%ax
0x8049eef nop
0x8049ef0 push %ebp
0x8049ef1 push %edi
0x8049ef2 push %esi
0x8049ef3 call 0x804aa73
0x8049ef8 add $0xb8108,%esi
0x8049efe push %ebx
download tmux:
sudo apt install tmux
Follow installation steps
type 'tmux' to open a shell
run binary './Ch18PackUnp_UnpackGdb'
click on the top left icon to open a new tab
In newly tabbed window type:
ps -ef | egrep Ch18
This will show process id's, you want the first number after your username for the process ./Ch18… type:
gdb attach *your process id #*
lay asm, gdb should be deep into the programs scanf (5 calls downward)
To get back up, you need to type:
finish
finish
finish
Now you should see the same code as the screen shot
type 'si' until you reach the highlighted code.
follow the command below to get PW.
native process 16077 In: L?? PC: 0x8049e9f
(gdb) finish
Run till exit from #0 0x08064adb in ?? ()
0x08051329 in ?? ()
(gdb) where
#0 0x08051329 in ?? ()
#1 0x0804a72d in ?? ()
(gdb) si
0x0805132c in ?? ()
0x08049e91 in ?? ()
(gdb) si
0x08049e94 in ?? ()
0x08049e97 in ?? ()
0x08049e9a in ?? ()
0x08049e9b in ?? ()
0x08049e9e in ?? ()
0x08049e9f in ?? ()
(gdb) x/s $ebp-0x20
0xffd57218: "TYMwNjI3"
(gdb)


jonharr2@jonharr2-VirtualBox: ~/metactf/Ch18-21
0x555555555452 <main+230> callq 0x5555555551c9 <print_msg
0x555555555457 <main+235> lea 0xba9(%rip),%rdi #
0x55555555545e <main+242> mov $0x0,%eax
0x555555555463 <main+247> callq 0x5555555550c0 <printf@pl
0x555555555468 <main+252> lea -0x14(%rbp),%rax
0x55555555546c <main+256> mov %rax,%rsi
0x55555555546f <main+259> lea 0xba6(%rip),%rdi #
0x555555555476 <main+266> mov $0x0,%eax
0x55555555547b <main+271> callq 0x5555555550d0 <__isoc99_
0x555555555480 <main+276> lea -0x14(%rbp),%rdx
0x555555555484 <main+280> lea -0x1d(%rbp),%rax
0x555555555488 <main+284> mov %rdx,%r9
0x55555555548b <main+287> mov %rax,%r8
0x55555555548e <main+290> lea 0xb8c(%rip),%rcx #
0x555555555495 <main+297> lea 0xb8e(%rip),%rdx #
0x55555555549c <main+304> lea 0xb90(%rip),%rsi #
0x5555555554a3 <main+311> lea 0xb92(%rip),%rdi #
B+>0x5555555554aa <main+318> callq 0x5555555552c7 <foo>
0x5555555554af <main+323> test %eax,%eax
0x5555555554b1 <main+325> je 0x5555555554c1 <main+341>
0x5555555554b3 <main+327> lea 0xb8b(%rip),%rdi #
0x5555555554ba <main+334> callq 0x555555555090 <puts@plt>
0x5555555554bf <main+339> jmp 0x5555555554cd <main+353>
0x5555555554c1 <main+341> lea 0xb87(%rip),%rdi #
0x5555555554c8 <main+348> callq 0x555555555090 <puts@plt>
0x5555555554cd <main+353> mov $0x0,%eax
0x5555555554d2 <main+358> mov -0x8(%rbp),%rcx
0x5555555554d6 <main+362> xor %fs:0x28,%rcx
0x5555555554df <main+371> je 0x5555555554e6 <main+378>
0x5555555554e1 <main+373> callq 0x5555555550b0 <__stack_c
0x5555555554e6 <main+378> leaveq
0x5555555554e7 <main+379> retq
0x5555555554e8 nopl 0x0(%rax,%rax,1)
0x5555555554f0 <__libc_csu_init> endbr64
0x5555555554f4 <__libc_csu_init+4> push %r15
0x5555555554f6 <__libc_csu_init+6> lea 0x289b(%rip),%r15
0x5555555554fd <__libc_csu_init+13> push %r14
native process 3424 In: main L?? PC: 0x5555555554aa
n this level, you will need to use gdb to find the password as it is
being passed as a parameter to a function with 6 parameters. For x86-64,
the mnemonic for passing parameters using registers is Diane's Silk Dress
Cost $89. (rdi rsi rdx rcx r8 r9)metactf/Ch18-21/Ch21x64_ParamsRegs
3
Enter the password:
(gdb) x/s $rcx0x00005555555554aa in main ()
0x555555556021: "Kr3Pp6kr"
(gdb) x/s $rdx
0x55555555602a: "uyQMBeoD"
(gdb) x/s $rsi
0x555555556033: "Ys1QXVxq"
(gdb) x/s $r8
0x7fffffffdf73: "YjNhYTEz"
(gdb)
b *main+318
x/s $r8

2. Ch21x64_ParamsStack

a. Open in Gdb
b. Find line in main after scanf and create breakpoint on this line. *main+168 for me
c. Run program and enter bogus password foo
d. When you hit the breakpoint si, single step, to each push %rax instruction and one at a time x/s the contents of $rax until you find the right password.