

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_Conditionals](#)
[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](#)

Structures

```

; Attributes: bp-based frame fuzzy-sp
; int __cdecl main(int argc, const char **argv, const char **envp)
public main
main proc near

var_1C= dword ptr -1Ch
var_15= byte ptr -15h
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, ecx
mov   eax, [eax+4]
mov   [ebp+var_1C], eax
mov   eax, large gs:14h
mov   [ebp+var_C], 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+var_15]
push  eax
push  offset a8s      ; "%8s"
call  _isoc99_scant
add   esp, 10h
sub   esp, 8
push  offset aNtvkzjbm ; "NTVkZjBm"
lea   eax, [-12+var_15]
push  eax
call  _strcmp
add   esp, 10h
test  eax, eax
jz    short loc_8049314

```

Enums

You can find the answer by typing strings
Ch01StatA_Readelf

Or you can view in IDA like so.

```

graph TD
    Start[Main Function] --> Print1[_printf("Enter the password:")]
    Print1 --> Read1[_isoc99_scant("%8s")]
    Read1 --> Compare1[_strcmp("NTVkZjBm")]
    Compare1 -- JZ --> GoodJob[loc_8049314: puts("Good Job.")]
    GoodJob -- ADD esp, 10h --> Exit[Exit]
    Compare1 -- NOT JZ --> TryAgain[loc_8049324: puts("Try again.")]
    TryAgain -- ADD esp, 10h --> Read1
    TryAgain -- ADD esp, 10h --> StackFail[stack_chk_fail]

```

jonharr2@jonharr2-VirtualBox:~/metactf/Ch01-08\$ ltrace ./Ch03DynA_Ltrace

```

 libc_start_main(0x804929d, 1, 0xffffd0aae4, 0x8049390 <unfinished ...>
 strlen("In this level, you will be force"...) = 311
 printf("%s", "In this level, you will be force"...) = 311
 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_scant(0x804a020, 0xffffd0aa04, 0xffffd0aa38, 0x80492c6) Enter the password: <no return ...>
--- SIGWINCH (Window changed) ---
asdf
<... _isoc99_scant_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      ; "t8s"
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)

```

; Attributes: bp-based frame fuzzy-sp

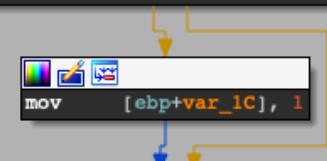
; int __cdecl main(int argc, const char **argv, const char **envp)
public main
main proc near

var_1D= byte ptr -1Dh
var_1C= dword ptr -1Ch
var_17= byte ptr -17h
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_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_17]
push  eax
push  offset a10s      ; "%10s"
call  _isoc99_scanf
add   esp, 10h
mov   [ebp+var_1D], 4Eh ; 'N'
movzx  eax, [ebp+var_17]
cmp   [ebp+var_1D], al
jz    short loc_80492FD

```

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

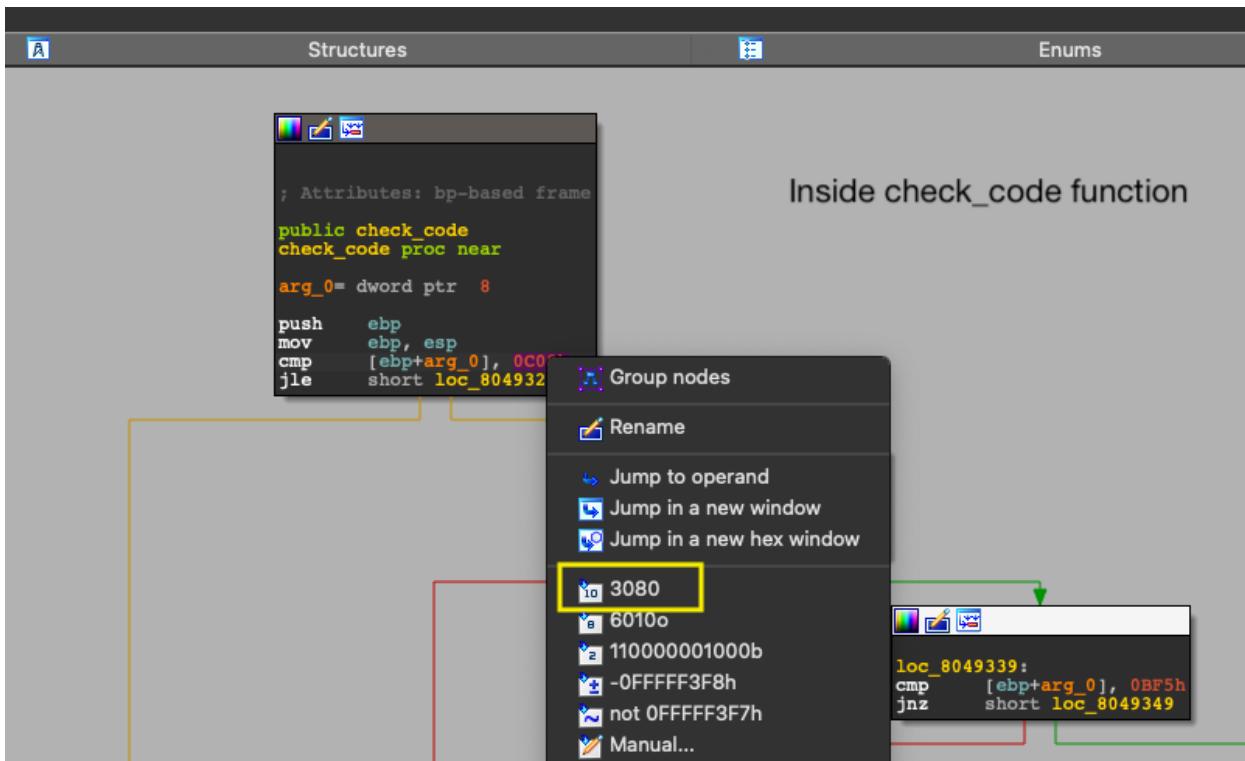


```

loc_80492FD:
lea   eax, [ebp+var_17]
add  eax, 1
sub  esp, 8
push offset aExmgyyzm ; "ExMGYyMzM"
push  eax
call _strcmp
add  esp, 10h
test eax, eax
jz   short loc_804931F

```





```
loc_8049339:  
cmp    [ebp+arg_0], 0BF5h  
jnz    short loc_8049349
```

```
; Attributes: bp-based frame
public loop_multi
loop_multi proc near

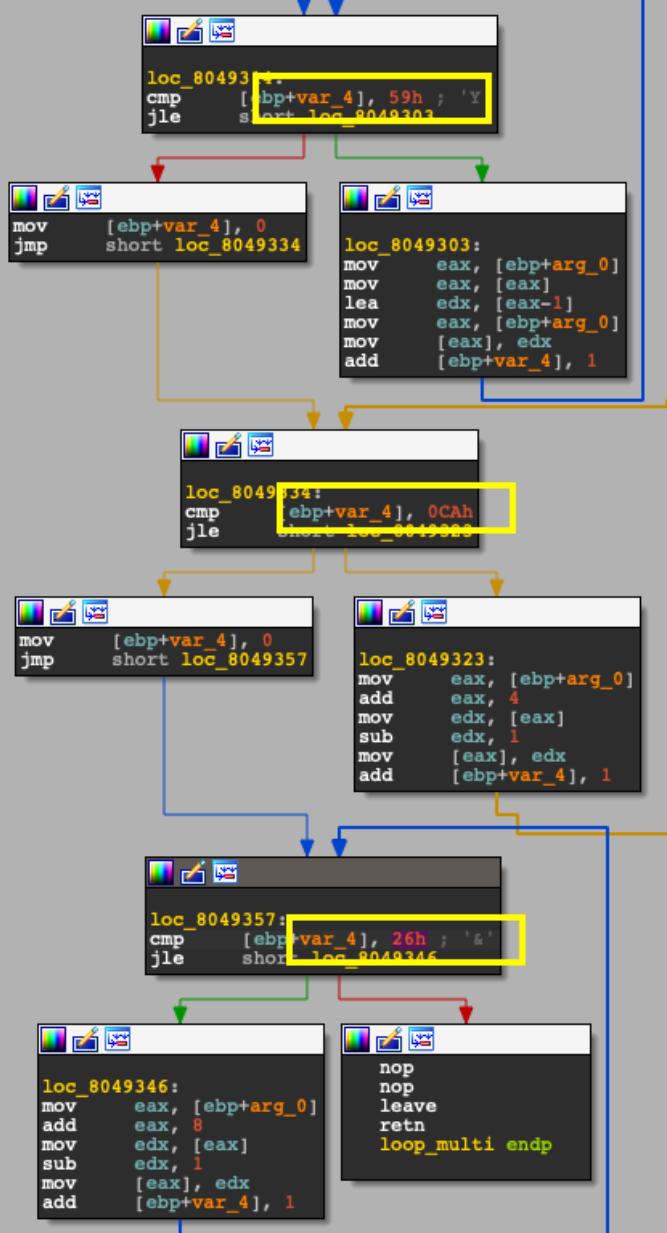
var_4= dword ptr -4
arg_0= dword ptr 8

push    ebp
mov     ebp, esp
sub     esp, 10h
mov     [ebp+var_4], 0
jmp     short loc_8049314
```

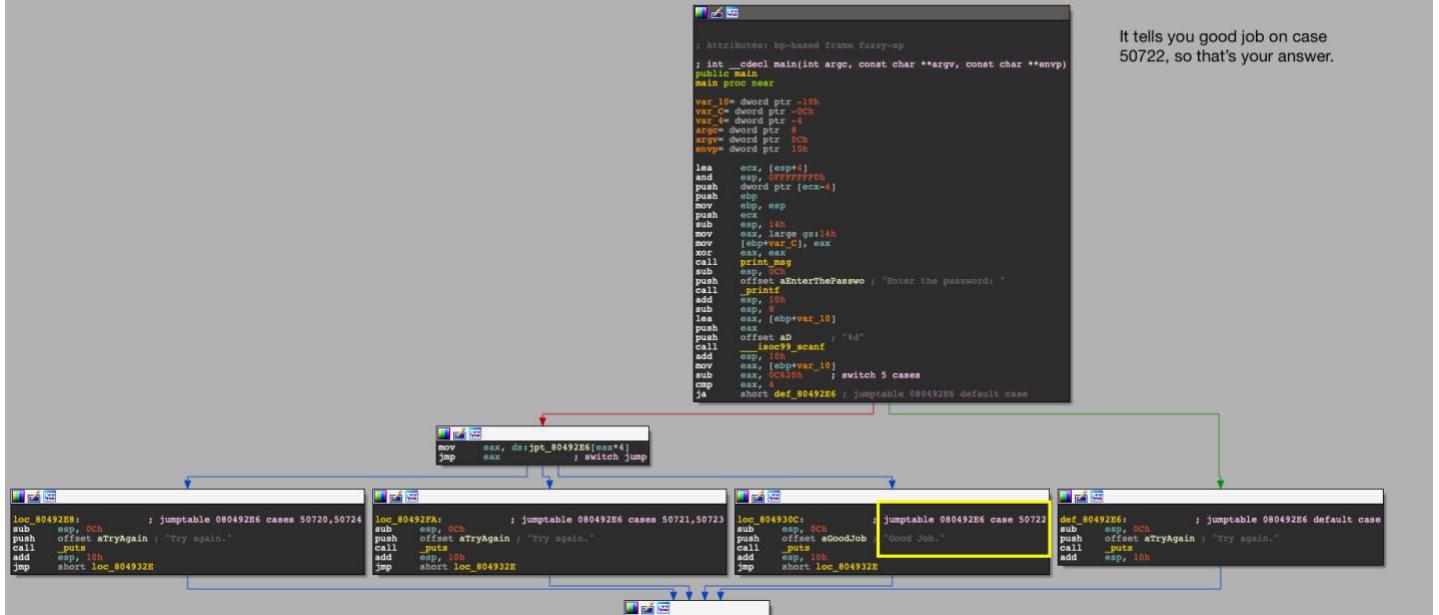
get the hex->decimal
of those highlighted
values

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"

```
; Attributes: bp-based frame fuzzy-sp
; int __cdecl main(int argc, const char **argv, const char **envp)
public main
main proc near

var_4C= dword ptr -4Ch
var_3C= dword ptr -3Ch
var_3B= dword ptr -38h
var_34= byte ptr -34h
var_20= byte ptr -20h
var_C= dword ptr -0Ch
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  ebx
push  ecx
sub   esp, 50h
mov   eax, ecx
mov   eax, [eax+4]
mov   [ebp+var_4C], eax
mov   eax, large gs:14h
mov   [ebp+var_C], eax
xor   eax, eax
mov   [ebp+var_38], 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_34]
push  eax
push  offset a8s      ; "$8s"
call  __isoc99_scantf
add   esp, 10h
mov   [ebp+var_3C], 0
jmp   short loc_8049360
```

```
loc_8049360:
mov   eax, [ebp+var_3C]
cmp   eax, 13h
jbe   short loc_804930B
```

```
push  offset avGbnuljg ; "VGbNuljG"
push  offset a8          ; "$8s"
push  9
lea   eax, [ebp+var_20]
push  eax
call  _snprintf
add   esp, 10h
sub   esp, 8
lea   eax, [ebp+var_20]
push  eax
lea   eax, [ebp+var_34]
push  eax
call  _strcmp
add   esp, 10h
test  eax, eax
jz    short loc_804940F
```

```
loc_804930B:
mov   eax, [ebp+var_3C]
mov   ebx, bogus[eax*4]
mov   eax, [ebp+var_3C]
mov   eax, bogus[eax*4]
sub   esp, 0Ch
push  eax
call  _strlen
add   esp, 10h
add   eax, 1
push  ebx
push  offset aS          ; "$s"
push  eax
lea   eax, [ebp+var_20]
push  eax
call  _snprintf
add   esp, 10h
sub   esp, 8
lea   eax, [ebp+var_20]
push  eax
lea   eax, [ebp+var_34]
push  eax
call  _strcmp
add   esp, 10h
test  eax, eax
jz    short loc_804935C
```

```

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
0xfffffd137:      "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
0xfffffd137:      '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.

```

; Attributes: bp-based frame fuzzy-sp
; int __cdecl main(int argc, const char **argv, const char **envp)
public main
main proc near

var_2C= dword ptr -2Ch
var_24= dword ptr -24h
var_20= byte ptr -20h
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, 34h
mov   eax, ecx
mov   eax, [eax+4]
mov   [ebp+var_2C], eax
mov   eax, large gs:14h
mov   [ebp+var_C], eax
xor   eax, eax
call  print_msg
push  esp, 0Ch
offset pwd      ; "YTZhYzhl"
add   esp, 10h
sub   esp, 0Ch
push  offset aEnterThePasswo ; "Enter the password: "
call  _printf
add   esp, 10h
sub   esp, 8
lea   eax, [ebp+var_20]
push  eax
push  offset a19s     ; "%19s"
call  __isoc99_scanf
add   esp, 10h
sub   esp, 0Ch
push  offset pwd      ; "YTZhYzhl"
call  _strlen
add   esp, 10h
mov   [ebp+var_24], eax
sub   esp, 0Ch
lea   eax, [ebp+var_20]
push  eax
call  _strlen
add   esp, 10h
mov   edx, [ebp+var_24]
cmp   eax, edx
jz    short loc_8049351

```

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

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

```

loc_8049351:
sub   esp, 8
push  offset pwd      ; "YTZhYzhl"
lea   eax, [ebp+var_20]
push  eax
call  _strcmp
add   esp, 10h
test  eax, eax
jz    short loc_8049380

```

```

tryAgain ; "Try again."
_8049395

```

```

sub   esp, 0Ch
push  offset aTryAgain ; "Try again."
call  __puts
add   esp, 10h
mov   eax, 0
jmp   short loc_8049395

```

```

loc_8049380:
sub   esp, 0Ch
push  offset aGoodJob ; "Good Job."
call  __puts
add   esp, 10h
mov   eax, 0

```

```

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   $0xfffffffff0,%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>

native_process 2127 In: retval_in_rax
2      breakpoint  keep y  0x08049341 <retval_in_rax+52>          L??   PC: 0x8049341
(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
0xfffffd13c:  "5\242Xe5\240\004\bT\321\377\377h\321\377\377\232\223\004\b\001"
(gdb) x /s $ebp-0x10
0xfffffd138:  "\n"
(gdb) p $eax
$1 = 1700307509
(gdb) 
```

```

Register group: general
eax          0x1          1
ecx          0x0          0
edx          0xfffffd138      -11976
ebx          0x0          0
esp          0xfffffd130      0xfffffd130
ebp          0xfffffd168      0xfffffd168
esi          0xf7fb2000      -134537216
edi          0xf7fb2000      -134537216
eip          0x804973d      0x804973d <main+110>
eflags        0x286        [ PF SF IF ]
cs           0x23         35      Set breakpoint at *main+110
ss           0x2b         43      type 'set {int*} 0xfffffd134 = 0x70
ds           0x2b         43      (0xfffffd134 is the address for $ebp-0x34)
es           0x2b         43      This will get you in the print_psw function.
fs           0x0          0

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
0xfffffd134:    "\001"
(gdb) x/xw $ebp-0x34
0xfffffd134:    0x00000001
(gdb) set {int*} 0xfffffd134 = 0x70
(gdb) x/xw $ebp-0x34
0xfffffd134:    0x00000070
(gdb) 

```

```

Register group: general
eax      0x4e          78      Set breakpoint at *test_c+67
ecx      0x59          89      x/xw $ebp+0xc gives you address
edx      0xfffffd0fe    -12034   x/s 0xffffd0fc dereferences the string at
ebx      0x0            0       the passwords address.
esp      0xfffffd070    0xfffffd070
ebp      0xfffffd098    0xfffffd098
esi      0xf7fb2000    -134537216
edi      0xf7fb2000    -134537216
eip      0x804925e     <test_c+67>
eflags   0x296         [ PF AF SF IF ]
cs       0x23          35

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
0xfffffd0a4: 0xfffffd0fc
(gdb) x/s $ebp+0xc
0xfffffd0a4: "\374\320\377\377\304\320\377\377"
(gdb) x/s $ebp+0x8
0xfffffd0a0: "\350\320\377\377\374\320\377\377\304\320\377\377"
(gdb) x/d $ebp+0x8
0xfffffd0a0: -24
(gdb) x/d $ebp+0xc
0xfffffd0a4: -4
(gdb) x/s 0xfffffd0fc
0xfffffd0fc: "YjNkxjUx"
(gdb) 

```

Register group: general

eax	0x80493c3	134517699
ecx	0x0	0
edx	0x443c	17468
ebx	0x0	0
esp	0xfffffd118	0xfffffd118
ebp	0xfffffd118	0xfffffd118
esi	0xf7fb2000	-134537216
edi	0xf7fb2000	-134537216
rip	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   $0xffffffff,%esp
 0x80492df <main+7>         pushl -0x4(%ecx)
 0x80492e2 <main+10>        push   %ebp
 0x80492e3 <main+11>        mov   %esp,%ebp
 0x80492e5 <main+13>        push   %ecx

```

L?? PC: 0x80492c5

```

native process 8518 In: check_code
(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
0xfffffd124: 33157
(gdb)

```

```

Register group: general
eax      0x80493a3    134517667    ecx      0x0        0
edx      0xf7fb2000   -134537216    ebx      0x0        0
esp      0xfffffd130   0xfffffd130    ebp      0xfffffd168  0xfffffd168
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)      Type: b *main+108
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

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
0xfffffd158:    88725
(gdb)

```

```

Register group: general
eax      0xfffffd114      -12012      ecx      0x80d4ea0      135089824
edx      0x8102000      135274496      ebx      0x8102000      135274496
esp      0xfffffd0e0      0xfffffd0e0      ebp      0xfffffd148      0xfffffd148
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
0x8049e92 <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\402\bf\220\377\8\020\bf\220\377<\020\bf\220\377@\020\bf\220\377\0\020\bf\220\203\35
4\bf\377t\$ \350\224,\002"
(gdb) x/s $ebp-0x34
0xfffffd114:    "2342"
(gdb) x/s $ebp-0x20
0xfffffd128:    "MzY02GZjYTL"
(gdb)

```

```

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      1. Datatype of scanf found here (%3d %3s%3x)
0x80493eb <main+334>    push   $0x804a020
0x80493f0 <main+339>    call   0x80490a0 <__isoc99_scanf@plt> x/s 0x804a020
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> First and last PW are tested here
0x8049400 <main+355>    sub    $0x8,%esp
0x8049403 <main+358>    lea    -0x10(%ebp),%eax (The first pw comes in a hex form and needs
0x8049406 <main+361>    push   %eax to be converted to decimal while the last PW is
0x8049407 <main+362>    lea    -0x14(%ebp),%eax already in hex so it stays the same.)
0x804940a <main+365>    push   %eax 'x2wx $ebp-0x1c'
B+>0x804940a <main+365>    push   %eax
0x804940b <main+366>    call   0x8049040 <strcmp@plt> The second PW is a string so it must go through a strcmp
0x8049410 <main+371>    add    $0x10,%esp set breakpoint at 'main+365', then examine the string:
0x8049413 <main+374>    test   %eax,%eax 'x/s $ebp-10'
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>

native process 8229 In: main
(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) ■

```

johnharr2@johnharr2-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: fffffd0e8
Segmentation fault. Try again.

johnharr2@johnharr2-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.

johnharr2@johnharr2-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.

johnharr2@johnharr2-VirtualBox:~/metactf/Ch01-08\$

Register group: general

eax	0x804b880	134527104
ecx	0x0	0
edx	0x804b880	134527104
ebx	0x0	0
esp	0xfffffd0d0	0xfffffd0d0
ebp	0xfffffd0e8	0xfffffd0e8
esi	0x7fb2000	-134537216
edi	0x7fb2000	-134537216
eip	0x80493d1	0x80493d1 <cats+62>
eflags	0x212	[AF IF]
cs	0x23	35
ss	0xb	43
ds	0xb	43
es	0xb	43
fs	0x0	0

```
b* 0x80493d0 <cats+36> push $0x804a038
b* 0x80493d0 <cats+41> call 0x8049400 <printf@plt>
b* 0x80493d1 <cats+46> add $0x10,%esp
b* 0x80493d1 <cats+49> sub $0xc,%esp
b* 0x80493d1 <cats+52> push $0x0
b* 0x80493d1 <cats+54> call 0x8049080 <exit@plt>
B* 0x80493d1 <cats+59> mov -0xc(%ebp),%eax
B* 0x80493d1 <cats+62> mov 0x14(%eax),%eax
B* 0x80493d1 <cats+65> mov %eax,-0xc(%ebp)
B* 0x80493d1 <cats+68> cmpl $0x0,-0xc(%ebp)
B* 0x80493d0 <cats+72> jne 0x80493a2 <cats+15>
0x80493d0 <cats+74> call 0x80492cd <print_good>
0x80493d0 <cats+79> nop
0x80493d0 <cats+80> leave
0x80493d0 <cats+81> ret
```

native process 2186 In: cats
Which has no line number information.

L?? PC: 0x80493d1

Breakpoint 10, 0x80493ab in cats ()
Single stepping until exit from function cats, which has no line number information.

Breakpoint 11, 0x80493ad in cats ()
Single stepping until exit from function cats, which has no line number information.

Breakpoint 20, 0x80493ce in cats ()
Single stepping until exit from function cats, which has no line number information.

Breakpoint 4, 0x80493d1 in cats ()
(gdb) [REDACTED]

type 'b "cats+62'
The loop will execute 4 times until it segs faults on the 5th.
Right before it segs 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:

```
8049030 <.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)  
...
```

```
8049040 <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>
```

```
8049050 <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>
```

```
8049060 <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:

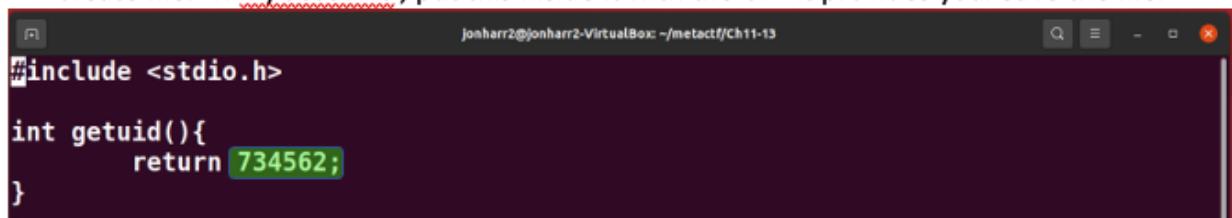
59578014 59575b96

- Run the program, find the UID hint

```
~/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 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.



```
#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 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 easiest.

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: Sm0XUCqj3
```

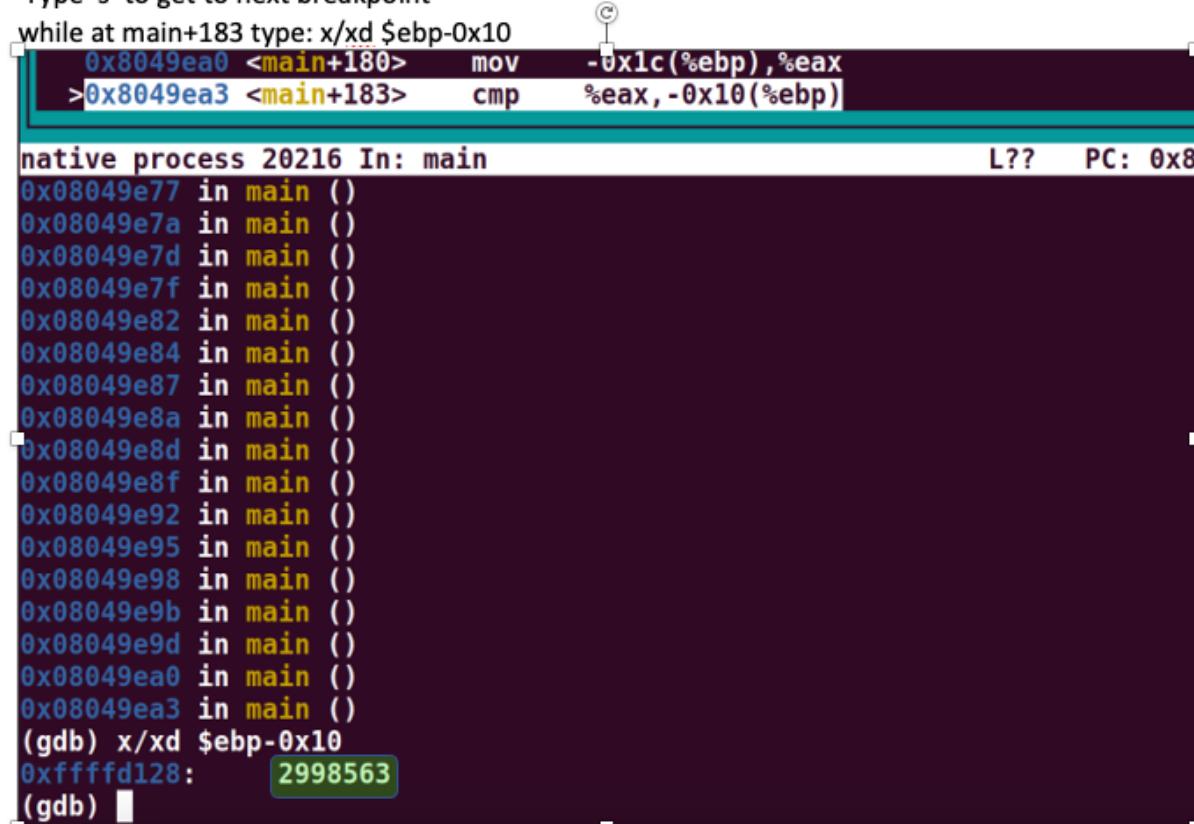
- Output should produce three zeros

- 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



The screenshot shows the GDB assembly window. At the top, there is assembly code:

```

0x8049ea0 <main+180>    mov    -0x1c(%ebp),%eax
>0x8049ea3 <main+183>    cmp    %eax,-0x10(%ebp)

```

A call instruction is highlighted with a yellow bracket and a circled 'C' icon above it.

Below the assembly code, the register dump shows:

```

native process 20216 In: main
L?? PC: 0x8
0x08049e77 in main ()
0x08049e7a in main ()
0x08049e7d in main ()
0x08049e7f in main ()
0x08049e82 in main ()
0x08049e84 in main ()
0x08049e87 in main ()
0x08049e8a in main ()
0x08049e8d in main ()
0x08049e8f in main ()
0x08049e92 in main ()
0x08049e95 in main ()
0x08049e98 in main ()
0x08049e9b in main ()
0x08049e9d in main ()
0x08049ea0 in main ()
0x08049ea3 in main ()
(gdb) x/xd $ebp-0x10
0xffffd128: 2998563
(gdb) █

```

The value at address `0xffffd128` is highlighted in a red box and shows the value `2998563`.

- 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`

```

0x804940d <try_command+199>    call  0x8049040 <read@plt>
B+>0x8049412 <try_command+204> add   $0x10,%esp
0x8049415 <try_command+207>    movl  $0x0,-0xe8(%ebp)
0x804941f <try_command+217>    jmp   0x804947b <try_command+309>
0x8049421 <try_command+219>    lea    -0xd4(%ebp),%edx
0x8049427 <try_command+225>    mov   -0xe8(%ebp),%eax
0x804942d <try_command+231>    add   %edx,%eax
0x804942f <try_command+233>    movzbl (%eax),%eax
0x8049432 <try_command+236>    test  %al,%al
0x8049434 <try_command+238>    je    0x804946b <try_command+293>
0x8049436 <try_command+240>    lea   -0x70(%ebp),%edx
0x8049439 <try_command+243>    mov   -0xe8(%ebp),%eax
0x804943f <try_command+249>    add   %edx,%eax
0x8049441 <try_command+251>    movzbl (%eax),%edx
0x8049444 <try_command+254>    lea   -0xd4(%ebp),%ecx
0x804944a <try_command+260>    mov   -0xe8(%ebp),%eax
0x8049450 <try_command+266>    add   %ecx,%eax
0x8049452 <try_command+268>    movzbl (%eax),%eax
0x8049455 <try_command+271>    cmp   %al,%dl
0x8049457 <try_command+273>    jne   0x8049462 <try_command+284>
0x8049459 <try_command+275>    movb  $0x1,-0xed(%ebp)

native process 20746 In: try command                                L??  PC: 0x80494
[Detaching after fork from parent process 20745]
[Inferior 1 (process 20745) detached]
[Switching to process 20746]
0x08049110 in close@plt ()
(gdb) s
Single stepping until exit from function close@plt,
which has no line number information.
0xf7ebb100 in close () from /lib32/libc.so.6
(gdb) s
Single stepping until exit from function close,
which has no line number information.
0x080493cc in try_command ()
(gdb) s
Single stepping until exit from function try_command,
which has no line number information.

Thread 2.1 "Ch12Covert_Fork" hit Breakpoint 1, 0x08049412 in try_command ()
(gdb) x/s $ecx
0xfffffd0e8:     "2VhMzc4Y"
(gdb) 

```

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

```

Enter the password: ^C
jonharr2@jonharr2-VirtualBox:~/metactf/Ch11-13$ ls
Ch11MalBeh_HijackPLT      Ch11MalBeh_LdPreloadRand   Ch12Covert_ForkPipe
Ch11MalBeh_HijackPLT.c    Ch11MalBeh_NetcatShovel   Ch13DataEnc_BaseEnc
Ch11MalBeh_LdPreloadGetUID Ch12Covert_ForkFollow   Ch13DataEnc_XorEnc
jonharr2@jonharr2-VirtualBox:~/metactf/Ch11-13$ strings ./Ch13DataEnc_BaseEnc
tD
/lib/ld-linux.so.2
libc.so.6
__IO_stdin_used
strncmp
__isoc99_scanf
puts
__stack_chk_fail
printf
strlen
malloc
__libc_start_main
free
GLIBC_2.7
GLIBC_2.4
GLIBC_2.0
__gmon_start__
[^]
Enter the password:
%15s
MWJhZTkw
Good Job.
Try again.
;*2$"

```

Encode to Base64 format

Simply enter your data then push the encode button.

① To encode binaries (like images, documents, etc.) use the file upload form a little further down on this page.

UTF-8 Destination character set.

LF (Unix) Destination newline separator.

Encode each line separately (useful for when you have multiple entries).

Split lines into 76 character wide chunks (useful for MIME).

Perform URL-safe encoding (uses Base64URL format).

Live mode OFF Encodes in real-time as you type or paste (supports only the UTF-8 character set).

> ENCODE < Encodes your data into the area below.

TVdKaFpUa3c=

```

Register group: general
eax          0x804c160      134529376
ecx          0x0            0
edx          0x41          65
ebx          0x0            0
esp          0xfffffd130    0xfffffd130
ebp          0xfffffd148    0xfffffd148
esi          0xf7fb2000    -134537216
edi          0xf7fb2000    -134537216
eip          0x804930e     0x804930e <main+145>
eflags        0x206         [ PF IF ]
cs           0x23          35      If you get the strings for the binary
ss           0x2b          43      it will tell you a long numeric value
ds           0x2b          43      It Xors that value with the hex value
es           0x2b          43      at line main+77, the result of that
fs           0x0            0       xor is what is stored into $eax at line
gs           0x63          99      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> To get the
0x8049303 <main+134>   mov    -0x10(%ebp),%eax      password go to:
0x8049306 <main+137>   add    $0x804c160,%eax
B+ 0x804930b <main+142> movzbl (%eax),%edx
>0x804930e <main+145> mov    -0x10(%ebp),%eax      b *main+145
0x8049311 <main+148>   add    $0x804c170,%eax      x/s $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

native process 5073 In: main
(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) q

```

- 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

The figure shows two side-by-side windows of the IDA Pro debugger. The left window, titled 'IDA View A', displays assembly code for a procedure named 'check_code'. The right window, titled 'IDA View B', displays assembly code for another procedure. Both windows show various assembly instructions like push, mov, and call, along with their corresponding opcodes and addresses.

IDA View A Context Menu:

- Renome
- Jump to address... (G)
- Mark position... (Alt+M)
- Edit function... (Alt+P)
- Hide
- Graph view
- Undefine
- Synchronize with

IDA View B Context Menu:

- dword ptr -14h
- dword ptr -0Ch
- push ebp
- mov esp, esp
- sub esp, 18h
- mov eax, large gs:14h
- mov [ebp+var_C], eax
- xor eax, eax
- sub esp, 0Ch
- push offset aEnterThePasswo ; "Enter"
 - _printf
- add esp, 10h
- sub esp, 8
- lea eax, [ebp+var_14]
- push eax
- push offset aD ; "%d"
- call __isoc99_scanf
- add esp, 10h
- call near ptr loc_8049328+3
- push ds

IDA View A Labels:

- check_code
- var_14
- var_C
- aEnterThePasswo
- aD
- __isoc99_scanf
- loc_8049328
- loc_8049329
- loc_804932A
- unk_804932B
- loc_8049342
- loc_8049342
- print_again
- loc_8049347
- loc_8049347
- print_again
- loc_8049347
- loc_8049347
- loc_8049347

IDA View B Labels:

- var_14
- var_C
- push
- ebp
- esp
- sub
- esp, 18h
- mov
- eax, large gs:14h
- mov
- [ebp+var_C], eax
- xor
- eax, eax
- sub
- esp, 0Ch
- push
- offset aEnterThePasswo
- call
- _printf
- add
- esp, 10h
- sub
- esp, 8
- lea
- eax, [ebp+var_14]
- push
- eax
- push
- offset aD
- call
- __isoc99_scanf
- add
- esp, 10h
- call
- near ptr unk_804932B
- push
- ds
- db 10h
- db 0Fh ; E
- db 0
- unk_804932B
- loc_8049342
- lock
- mov eax, [ebp-10h]
- mov
- edx, [eax]
- mov
- eax, [ebp-14h]
- cpx
- edx, eax
- jnz
- short loc_8049342
- call
- print_good
- jnp
- short loc_8049347
- call
- print_again
- sub
- esp, 0Ch
- push
- 8

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.



```
; int __cdecl main(int argc, const char **argv, const char **envp
public main
main:
    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
    push  eax
    cmp   eax, eax
    jz    short near ptr loc_8049344+
.text:08049344 loc_8049344+
    .Renam
    ↪ Jump to operand
    ↪ Jump in a new window
    ↪ Jump in a new hex window
    List cross references to...
    ↪ Create function...
    ✘ Undefine
```

```
; int __cdecl main(int argc, const char **argv, const char **envp
public main
main:
    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
    push  eax
    cmp   eax, eax
    jz    short loc_8049345
.text:08049344
.text:08049345
loc_8049345:
    pop   eax
    mov   dword ptr [ebp-10h], 0EB80CD8
    sub   esp, 0Ch
    push  offset aEnterThePasswo ; "Enter the password"
    call  printf
```

- 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

```

.4000:00049350 ; 
.text:08049352      db 0Fh
.4000:00049353 ; 
.4000:00049353 loc_8049353:    mov    dword ptr [ebp-10h], 0C500B9h
.4000:00049354     sub    esp, 0Ch
.4000:00049355     push   offset aEnterThePasswo ; "Enter the password: "
.4000:00049356     call   _printf
.4000:00049357     add    esp, 10h
.4000:00049358     sub    esp, 8
.4000:00049359     lea    eax, [ebp-14h]
.4000:0004935A     push   eax
.4000:0004935B     offset aD           ; "%d"
.4000:0004935C     call   __isoc99_scanf
.4000:0004935D     add    esp, 10h
.4000:0004935E     stc
.4000:0004935F     cmc
.4000:00049360     jnb   short near ptr loc_8049382+1
.4000:00049361
.4000:00049362 loc_8049382:    jnb   near ptr 683CDh
.4000:00049363     jnb   short near ptr loc_804938A+1
.4000:00049364
.4000:00049365 loc_804938A:    jnb   near ptr 8B0683D5h
.4000:00049366     in    al, dx
.4000:00049367     or    al, 6Ah
.4000:00049368     add   eax, ebp
.4000:00049369     mov    bh, 0FCh
.4000:0004936A ;

```

- 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

```

.4000:00049350 ; 
.text:08049352      db 0Fh
.4000:00049353 ; 
.4000:00049353 loc_8049353:    mov    dword ptr [ebp-10h], 0C500B9h
.4000:00049354     sub    esp, 0Ch
.4000:00049355     push   offset aEnterThePasswo ; "Enter the password: "
.4000:00049356     call   _printf
.4000:00049357     add    esp, 10h
.4000:00049358     sub    esp, 8
.4000:00049359     lea    eax, [ebp-14h]
.4000:0004935A     push   eax
.4000:0004935B     offset aD           ; "%d"
.4000:0004935C     call   __isoc99_scanf
.4000:0004935D     add    esp, 10h
.4000:0004935E     stc
.4000:0004935F     cmc
.4000:00049360     jnb   short loc_8049383
.4000:00049361
.4000:00049362 loc_8049383:    db 0Fh
.4000:00049363
.4000:00049364 loc_8049384:    add   dword ptr [ebp-10h], 1
.4000:00049365
.4000:00049366 loc_8049385:    clc
.4000:00049367     jnb   short loc_804938B
.4000:00049368
.4000:00049369 loc_804938B:    db 0Fh
.4000:00049370
.4000:00049371 loc_804938C:    add   dword ptr [ebp-10h], 1
.4000:00049372     sub    esp, 0Ch
.4000:00049373     push   1
.4000:00049374     call   _sleep
.4000:00049375     add    esp, 10h
.4000:00049376     clc
.4000:00049377     cmc
.4000:00049378     jb    short loc_80493A1
.4000:00049379
.4000:00049380 loc_80493A0:    db 0Fh
.4000:00049381
.4000:00049382 loc_804938B:    mov    eax, [ebp-14h]
.4000:00049383     cmp    [ebp-10h], eax
.4000:00049384     jnz   short loc_80493B0
.4000:00049385     call   print_good
.4000:00049386     jmp   short loc_80493B5
.4000:00049387

```

- 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

```

.text:00049310 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00049314
.text:00049318 public main
.text:0004931C
main:0004931E           lea    ecx, [esp+4]          ; DATA XREF: _start+2A+0
main:00049320           and   esp, 0FFFFFFF0h
main:00049322           push  dword ptr [ecx-4]
main:00049324           push  ebp
main:00049326           mov   ebp, esp
main:00049328           push  ecx
main:0004932A           sub   esp, 24h
main:0004932C           mov   eax, ecx
main:0004932E           mov   eax, [eax+4]
main:00049330           mov   [ebp-1Ch], eax
main:00049332           mov   eax, large gs:14h
main:00049334           mov   [ebp-0Ch], eax
main:00049336           xor   eax, eax
main:00049338           call  print_msg
main:0004933A           sub   esp, 0Ch
main:0004933C           push  offset aEnterThePasswo ; "Enter the password: "
main:0004933E           call  _printf
main:00049340           add   esp, 10h
main:00049342           sub   esp, 8
main:00049344           lea   eax, [ebp-18h]
main:00049346           push  eax
main:00049348           push  offset ad ; "%d"
main:0004934A           call  __isccc99_scantf
main:0004934C           add   esp, 10h
main:0004934E           mov   dword ptr [ebp-14h], 0A90AACh
main:00049350           mov   eax, 2

main:00049354F loc_804936F: ; CODE XREF: .text:loc_804936F+j
main:000493550           jmp   short near ptr loc_804936F+1
main:000493552 ;
main:000493554           sar   bh, 0C0h
main:000493556           mov   [ebp-10h], eax
main:000493558           mov   eax, [ebp-10h]
main:000493560           mov   ecx, eax
main:000493562           shl   dword ptr [ebp-14h], cl
main:000493564           mov   eax, [ebp-18h]
main:000493566           cmp   [ebp-14h], eax
main:000493568           jnz   short loc_804938E
main:00049356A           call  print_good
main:00049356C           jmp   short loc_8049393

```

Bit Shift Calculator

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

<div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;">Number Base</div> <div style="border: 1px solid #ccc; padding: 2px; width: 150px; display: inline-block;">Hexadecimal (16)</div> <div style="border: 1px solid #ccc; padding: 2px; width: 150px; display: inline-block;">Number</div> <div style="border: 1px solid #ccc; padding: 2px; width: 150px; display: inline-block;">A90AAC</div> <div style="border: 1px solid #ccc; padding: 2px; width: 150px; display: inline-block;">Shift Direction</div> <div style="border: 1px solid #ccc; padding: 2px; width: 150px; display: inline-block;"><< Left</div> <div style="border: 1px solid #ccc; padding: 2px; width: 150px; display: inline-block;">Bits to Shift</div> <div style="border: 1px solid #ccc; padding: 2px; width: 150px; display: inline-block;">4</div>	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 2px;">Decimal</td> <td style="padding: 2px; background-color: #e0f2e0;">177253056</td> </tr> <tr> <td style="padding: 2px;">Binary</td> <td style="padding: 2px;">1010100100001010101011000000</td> </tr> <tr> <td style="padding: 2px;">Hexadecimal</td> <td style="padding: 2px;">a90aac0</td> </tr> <tr> <td style="padding: 2px;">Octal</td> <td style="padding: 2px;">1244125300</td> </tr> </table>	Decimal	177253056	Binary	1010100100001010101011000000	Hexadecimal	a90aac0	Octal	1244125300
Decimal	177253056								
Binary	1010100100001010101011000000								
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 $1258796 + 5 = 1258801$

```

.text:080492F0 _checkit:                                ; CODE XREF: main+57+j
.text:080492F0                                         ; DATA XREF: main+52+o
    .text:080492F0         nop
    .text:080492F1         mov    eax, ds:check
    .text:080492F6         add    eax, 5
    .text:080492F9         mov    ds:check, eax
    .text:080492FB         push   offset _transpose
    .text:08049303         retn
    .text:08049303 _check_it      endp
    .text:08049303
    .text:08049304         db    90h
    .text:08049305         ; -----
    .text:08049305         pop    ebp
    .text:08049306         retn
    .text:08049307
    .text:08049307         ; ----- SUBROUTINE -----
    .text:08049307
    .text:08049307         Attributes: bp-based frame
    .text:08049307
    .text:08049307         public detectTrace
    .text:08049307 detectTrace     proc near             ; CODE XREF: __libc_csu_init+4C+p
    .text:08049307                                         ; DATA XREF: .init_array:_frame_dur
    .text:08049307         push   ebp
    .text:08049308         mov    ebp, esp
    .text:0804930A         sub    esp, 8
    .text:0804930D         push   0
    .text:0804930F         push   1
    .text:08049311         push   0
    .text:08049313         push   0
    .text:08049315         call   _ptrace
    .text:0804931A         add    esp, 10h
    .text:0804931D         test   eax, eax
    .text:0804931F         jns    short loc_804933B
    .text:08049321         sub    esp, 0Ch
    .text:08049324         push   offset aSorryWeHaveDis ; "Sorry, we have disallowed"
    .text:08049329         call   _puts
    .text:0804932E         add    esp, 10h
    .text:08049331         sub    esp, 0Ch
    .text:08049334         push   1
    .text:08049336         call   _exit
    .text:0804933B loc_804933B:                      ; CODE XREF: detectTrace+18+j
    .text:0804933B         nop
    .text:0804933C         leave
    .text:0804933D         retn
    .text:0804933D detectTrace     endp
    .text:0804933D
    .text:0804933E         ; ----- SUBROUTINE -----
    .text:0804933E
    .text:0804933E         Attributes: bp-based frame fuzzy-sp
    .text:0804933E
    .text:0804933E         ; int __cdecl main(int argc, const char **argv, const char **envp)
    .text:0804933E         public main
    .text:0804933E main          proc near             ; DATA XREF: _start+2A+o
    .text:0804933E
    .text:0804933E         argc          = dword ptr  8
    .text:0804933E         argv          = dword ptr  0Ch
    .text:0804933E         envp          = dword ptr  10h
    .text:0804933E
    .text:0804933E         lea    ecx, [esp+4]
    .text:08049342         and    esp, 0FFFFFF0h
    .text:08049345         push   dword ptr [ecx-4]
    .text:08049348         push   ebp
    .text:08049349         mov    ebp, esp
    .text:0804934B         push   ecx
    .text:0804934C         sub    esp, 4
    .text:0804934F         mov    ds:check, 13352Ch
    .text:08049359         call   print_msg
    .text:0804935E         sub    esp, 0Ch
    .text:08049361         push   offset aEnterInThePass ; "Enter in the password: "
    .text:08049366         call   _printf
    .text:0804936B         add    esp, 10h
    .text:0804936B         sub    esp, 4

```

```

jonharr2@jonharr2-VirtualBox: ~/metactf/Ch15-16
0x8049380 <main+147> incl -0x137cef3c(%ebx) open in gdb, Type:
0x8049386 <main+153> or $0x68,%al 'lay asm'
0x8049388 <main+155> das
0x8049389 <main+156> mov 0xbfe80804,%al b *0x804936C
0x804938e <main+161> cld b *0x80493B3
0x804938f <main+162> (bad) run
0x8049390 <main+163> incl -0x137cef3c(%ebx) You will get to first breakpoint
0x8049396 <main+169> or %cl,0x6850cc45(%ebp) set $eip = 0x8049376
0x804939c <main+175> inc %esp continue, enter some PW
0x804939d <main+176> mov 0x1be80804,%al at second bp type: x/2xw $esp
0x80493a2 <main+181> std This will give you two addresses
0x80493a3 <main+182> (bad) PW is in one of those addresses.
0x80493a4 <main+183> incl -0x137cef3c(%ebx) Follow commands below.
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
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
0xfffffd0e4: 0xfffffd114 0xfffffd128
(gdb) x/s 0xfffffd114
0xfffffd114: "foo"
(gdb) x/s 0xfffffd128
0xfffffd128: "cTM2NWI4"
(gdb) 

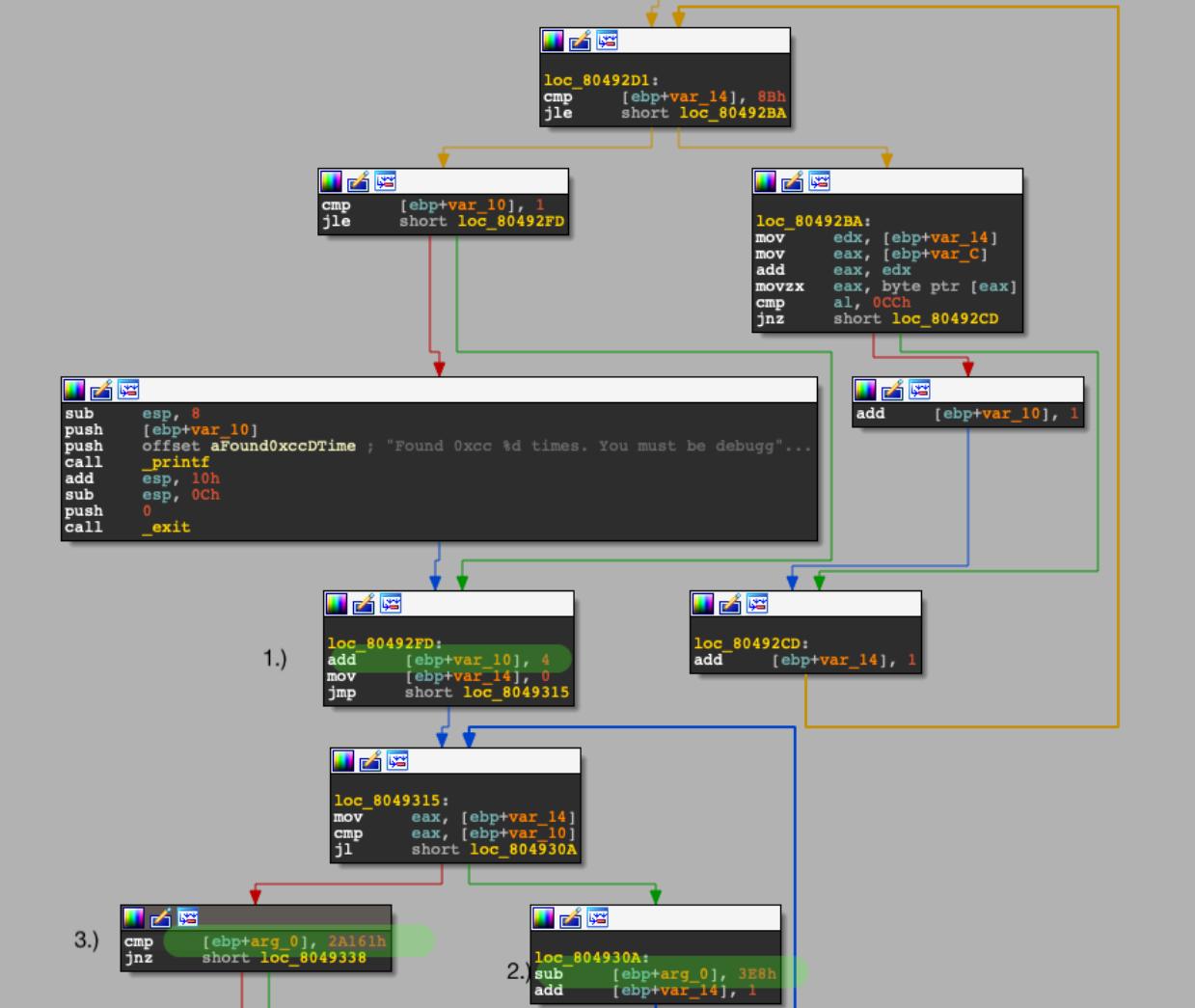
```

```
Jonharr2@jonharr2-VirtualBox: ~/metactf/Ch15-16
```

<pre>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</pre> <pre>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) ■</pre> <p style="text-align: right;">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</p>

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
 0xA161, 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
```



```

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_sccanf@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 ()                  Open in gdb
(gdb) set $eax = 0                                         b *sigtrap+42
(gdb) b *main+174                                         b *main+174
Breakpoint 3 at 0x80493f8                                 run (you will get sig trap alert)
(gdb) s                                                       continue until you hit first breakpoint
Single stepping until exit from function sigtrap,        set $eax = 0
which has no line number information.                      step (until you hit second bp)
0x080493e2 in main ()                                    x/s $ebp-0x34
(gdb) s                                                      
Single stepping until exit from function main,
which has no line number information.

Breakpoint 3, 0x080493f8 in main ()                      Open in gdb
(gdb) x/s $ebp-0x34                                     b *sigtrap+42
0xfffffd114:      "OWE1MjY4"                           b *main+174
(gdb)

```

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.

```
.text:080492DD .text:080492DD ; Attributes: bp-based frame
.text:080492DD
.text:080492DD public sigtrap
.text:080492DD proc near           ; CODE XREF: main+93↓p
.text:080492DD push    ebp
.text:080492DE mov     ebp, esp
.text:080492E0 sub     esp, 8
.text:080492E3 sub     esp, 8
.text:080492E6 push    offset sigtrap handler ; _sig_func_ptr
.text:080492EB push    5             ; int
.text:080492ED call    _signal
.text:080492F2 add    esp, 10h
.text:080492F5 sub    esp, 0Ch
.text:080492F8 push    5             ; int
.text:080492FA call    _raise
.text:080492FF add    esp, 10h
.text:08049302 nop
.text:08049303 leave
.text:08049304 retn
.text:08049304 sigtrap
.endp
.text:08049304
.text:08049305
.text:08049305 ; ===== S U B R O U T I N E =====
.text:08049305 .text:08049305 ; Attributes: bp-based frame
.text:08049305
.text:08049305 public print_good
.text:08049305 proc near           ; CODE XREF: main:loc_8049407↓p
```

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

```
.text:080493A6 sub    esp, 0Ch
.text:080493A9 push    offset aEnterThePasswo ; "Enter the password: "
.text:080493A9 call    _printf
.text:080493B1 add    esp, 10h
.text:080493B6 sub    esp, 8
.text:080493B9 lea    eax, [ebp+var_20]
.text:080493BC push    eax
.text:080493BD push    offset a19$ ; "%19$"
.text:080493C2 call    __isoc99_scanf
.text:080493C7 add    esp, 10h
.text:080493CA call    sigtrap
.text:080493CF mov    eax, myinteger
.text:080493D4 sub    esp, 4
.text:080493D7 push    eax, size_t
.text:080493D8 push    offset aNdhlytf1zg ; "Ndhlytf1zg"
.text:080493D9 lea    eax, [ebp+var_34]
.text:080493E0 push    eax, char *
.text:080493E1 call    _strncpy
.text:080493E6 add    esp, 10h
.text:080493E9 sub    esp, 8
.text:080493EC lea    eax, [ebp+var_34]
.text:080493F0 push    eax, char *
.text:080493F0 lea    eax, [ebp+var_20]
.text:080493F3 push    eax, char *
.text:080493F4 call    _strcmp
.text:080493F9 add    esp, 10h
.text:080493FC test   eax, eax
.text:080493FE jz    short loc_8049407
.text:08049400 call    print_again
.text:08049405 jmp    short loc_804940C
.text:08049407
.text:08049407 loc_8049407:           ; CODE XREF: main+C7?j
.text:08049407 call    print_good
.text:0804940C
.text:0804940C loc_804940C:           ; CODE XREF: main+4C?j
```

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.

The screenshot shows two side-by-side GDB sessions. Both sessions are running the same program, likely a debugger test, and both have stopped at the same point in the `sigtrap` function.

Left Session:

```

Register group: general
eax      0x1          1
ecx      0x0          0
edx      0xfffffb2000 -134537216
ebx      0x0          0
esp      0xfffffd148  0xfffffd148
ebp      0xfffffd158  0xfffffd158
esi      0x77fb2000 -134537216
edi      0x77fb2000 -134537216
rip      0x80492e6 0x80492e6 <sigtrap+9>
eflags   0x296        [ PF AF SF IF ]
cs       0x23         35
ss       0x2b         43

0x00000000 <sigtrap>    push  %rbp
0x00000000 <sigtrap+1>  mov   %rsp,%rbp
0x00000000 <sigtrap+3>  sub   $0x8,%rsp
0x00000003 <sigtrap+6>  sub   $0x8,%rsp
B+ 0x00000006 <sigtrap+9> push  $0x80492cd
b+ 0x00000009 <sigtrap+14> push  $0x5
b+ 0x0000000d <sigtrap+16> call  0x00000070 <signal@plt>
0x00000011 <sigtrap+21>  add   $0x10,%esp
0x00000015 <sigtrap+24>  sub   $0xc,%esp
0x00000019 <sigtrap+27>  push  $0x5
b+ 0x00000023 <sigtrap+29> call  0x00000000 <raise@plt>
b+ 0x00000027 <sigtrap+34> add   $0x10,%esp

native process 3117 Int: sigtrap
Program terminated with signal SIGSEGV, Segmentation fault.
The program no longer exists.
(gdb) run
Starting program: /home/hueb/Desktop/metactf/Ch16AntiDbg_SigtrapEntangle

Breakpoint 5, 0x0000003c in main ()
(gdb) s
Single stepping until exit from function main,
which has no line number information.

Breakpoint 9, 0x0000002e in sigtrap ()
(gdb) set $elp = 0x80492cd

```

Right Session:

```

Register group: general
eax      0x1          1
ecx      0x0          0
edx      0xfffffb2000 -134537216
ebx      0x0          0
esp      0xfffffd148  0xfffffd148
ebp      0xfffffd158  0xfffffd158
esi      0x77fb2000 -134537216
edi      0x77fb2000 -134537216
rip      0x80492cd 0x80492cd <sigtrap_handler>
eflags   0x296        [ PF AF SF IF ]
cs       0x23         35
ss       0x2b         43

b+ 0x0000002e <sigtrap_handler> push  %rbp
b+ 0x0000002f <sigtrap_handler+1> mov   %rsp,%rbp
b+ 0x00000030 <sigtrap_handler+3> movl $0x8,0x804c300
0x00000030 <sigtrap_handler+13> nop
0x00000030 <sigtrap_handler+14> pop   %rbp
0x00000030 <sigtrap_handler+15> ret
0x00000030 <sigtrap>    push  %rbp
0x00000030 <sigtrap+1>  mov   %rsp,%rbp
0x00000030 <sigtrap+3>  sub   $0x8,%rsp
0x00000030 <sigtrap+6>  sub   $0x8,%esp
B+ 0x00000030 <sigtrap+9> push  $0x80492cd
b+ 0x00000030 <sigtrap+14> push  $0x5

native process 3107 Int: sigtrap handler
Start it from the beginning? (y or n) y
Starting program: /home/hueb/Desktop/metactf/Ch16AntiDbg_SigtrapEntangle

Breakpoint 5, 0x0000003c in main ()
(gdb) s
Single stepping until exit from function main,
which has no line number information.

Breakpoint 9, 0x0000002e in sigtrap ()
(gdb) set $elp = 0x80492cd
Invalid cast.
(gdb) set $elp = 0x80492cd
(gdb) 
```

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

```
Register group: general
eax            0xfffffd174      -11916
ecx            0xf7e61340      -135916736
edx            0xa             10
ebx            0x0             0
esp            0xfffffd158      0xfffffd158
ebp            0xfffffd1a8      0xfffffd1a8
esi            0xf7fb2000      -134537216
edi            0xf7fb2000      -134537216
eip            0x80493ef       0x80493ef <main+184>
eflags          0x292          [ AF SF IF ]
cs             0x23           35
ss             0x2b           43

B+ 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
B+>0x80493ef <main+184>    push  %eax
0x80493f0 <main+185>    lea    -0x20(%ebp),%eax
b+ 0x80493f3 <main+188>    push  %eax
b+ 0x80493f4 <main+189>    call   0x8049050 <strcmp@plt>
0x80493f9 <main+194>    add    $0x10,%esp
b+ 0x80493fc <main+197>    test   %eax,%eax
0x80493fe <main+199>    je    0x8049407 <main+208>

native process 3127 In: main                                         L??
0xf7fe7ac4 in ?? () from /lib/ld-linux.so.2
0xf7fe7ac5 in ?? () from /lib/ld-linux.so.2
0xf7fe7ac6 in ?? () from /lib/ld-linux.so.2
0xf7fe7ac7 in ?? () from /lib/ld-linux.so.2
(gdb) c
Continuing.

Breakpoint 2, 0x080493ef in main ()
(gdb) p/s $eax
$60 = -11916
(gdb) x/s $eax
0xfffffd174:  "NDhiYTFI"
(gdb) █
```

```

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      in gdb set two breakpoints
0x80493d4 <main+157>   sub $0x4,%esp        b *0x80493bd
0x80493d7 <main+160>   push %eax          b *0x804936a
0x80493d8 <main+161>   push $0x804a03a      run, at first bp type
0x80493dd <main+166>   lea -0x34(%ebp),%eax  set $eip = 0x80492f3
0x80493e0 <main+169>   push %eax          step until you have to enter PW
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  This should take you to
Bb->0x80493ef <main+184> push %eax          second BP, enter command
0x80493f0 <main+185>   lea -0x20(%ebp),%eax shown below
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>

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
0xfffffd114: "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_sccanf@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)      Set breakpoints shown below
0x80493b6 <main+108>   mov    -0x10(%ebp),%edx      run, enter '100' for PW
0x80493b9 <main+111>   mov    0x804c15c,%eax      at first BP type:
B+>0x80493be <main+116> cmp    %eax,%edx      set $eip = 0x8049343
0x80493c0 <main+118>   jne   0x80493d4 <main+138> step until you get to main+116
0x80493c2 <main+120>   sub    $0xc,%esp      p/d $eax
0x80493c5 <main+123>   push   $0x804a037      p/d $edx
0x80493ca <main+128>   call   0x8049070 <puts@plt> edx was '100' and it was
0x80493cf <main+133>   add    $0x10,%esp      incremented by 5, the other
0x80493d2 <main+136>   jmp   0x80493e4 <main+154> number is the target it needs to
0x80493d4 <main+138>   sub    $0xc,%esp      get to so I did,
0x80493d7 <main+141>   push   $0x804a041      13591225 - 5 = 13591220
0x80493dc <main+146>   call   0x8049070 <puts@plt> That's the PW
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>

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
0x8049eee3 mov -0x4(%ebp),%ecx
0x8049eee6 leave
0x8049eee7 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 './Ch18PackUp_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
0xffffd57218:    "TYMwNjI3"
(gdb) 

```

```

jonharr2@jonharr2-VirtualBox: ~/metactf/Ch18-21
0x5555555555452 <main+230>           callq  0x55555555551c9 <print_msg
0x5555555555457 <main+235>           lea    0xba9(%rip),%rdi      #
0x555555555545e <main+242>           mov    $0x0,%eax
0x5555555555463 <main+247>           callq  0x55555555550c0 <printf@pl
0x5555555555468 <main+252>           lea    -0x14(%rbp),%rax
0x555555555546c <main+256>           mov    %rax,%rsi
0x555555555546f <main+259>           lea    0xba6(%rip),%rdi      #
0x5555555555476 <main+266>           mov    $0x0,%eax
0x555555555547b <main+271>           callq  0x55555555550d0 <_isoc99_
0x5555555555480 <main+276>           lea    -0x14(%rbp),%rdx
0x5555555555484 <main+280>           lea    -0x1d(%rbp),%rax
0x5555555555488 <main+284>           mov    %rdx,%r9
0x555555555548b <main+287>           mov    %rax,%r8
0x555555555548e <main+290>           lea    0xb8c(%rip),%rcx      #
0x5555555555495 <main+297>           lea    0xb8e(%rip),%rdx      #
0x555555555549c <main+304>           lea    0xb90(%rip),%rsi      #
0x55555555554a3 <main+311>           lea    0xb92(%rip),%rdi      #
B+>0x55555555554aa <main+318>         callq  0x55555555552c7 <foo>
0x55555555554af <main+323>           test   %eax,%eax
0x55555555554b1 <main+325>           je    0x5555555554c1 <main+341>
0x55555555554b3 <main+327>           lea    0xb8b(%rip),%rdi      #
0x55555555554ba <main+334>           callq  0x5555555555090 <puts@plt>
0x55555555554bf <main+339>           jmp    0x55555555554cd <main+353>
0x55555555554c1 <main+341>           lea    0xb87(%rip),%rdi      #
0x55555555554c8 <main+348>           callq  0x5555555555090 <puts@plt>
0x55555555554cd <main+353>           mov    $0x0,%eax
0x55555555554d2 <main+358>           mov    -0x8(%rbp),%rcx
0x55555555554d6 <main+362>           xor    %fs:0x28,%rcx
0x55555555554df <main+371>           je    0x5555555554e6 <main+378>
0x55555555554e1 <main+373>           callq  0x55555555550b0 <_stack_c
0x55555555554e6 <main+378>           leaveq
0x55555555554e7 <main+379>           retq
                                         nopl    0x0(%rax,%rax,1)
0x55555555554e8 <_libc_csu_init>     endbr64
0x55555555554f0 <_libc_csu_init+4>   push   %r15
0x55555555554f4 <_libc_csu_init+6>   lea    0x289b(%rip),%r15
0x55555555554fd <_libc_csu_init+13>  push   %r14
native process 3424 In: main          L?? PC: 0x55555555554aaI
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 $rcx0x0000555555554aa in main ()
0x555555556021: "Kr3Pp6kr"
(gdb) x/s $rdx
0x55555555602a: "uyQMBeoD"
(gdb) x/s $rsi
0x555555556033: "Ys1QXVxq"
(gdb) x/s $r8
0x7fffffffdf73: "YjNhYTEz"
(gdb) ■

```

2. Ch21x64_ParamsStack

- Open in Gdb
- Find line in main after scanf and create breakpoint on this line. *main+168 for me
- Run program and enter bogus password foo
- 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.

```
0x5555555553dd <main+113>    movb  $0x0,-0x15(%rbp)
Thunderbird Mail:5553e1 <main+117>    mov   $0x0,%eax
0x5555555553e6 <main+122>    callq 0x5555555551c9 <print_msg>
0x5555555553eb <main+127>    lea   0xc15(%rip),%rdi      # 0x555555556007
0x5555555553f2 <main+134>    mov   $0x0,%eax
0x5555555553f7 <main+139>    callq 0x5555555550c0 <printf@plt>
0x5555555553fc <main+144>    lea   -0x14(%rbp),%rax
0x555555555400 <main+148>    mov   %rax,%rsi
0x555555555403 <main+151>    lea   0xc12(%rip),%rdi      # 0x55555555601c
0x55555555540a <main+158>    mov   $0x0,%eax
0x55555555540f <main+163>    callq 0x5555555550de <_isoc99_scanf@plt>
B* 0x555555555414 <main+168>    lea   -0x14(%rbp),%rax
0x555555555418 <main+172>    push  %rax
0x555555555419 <main+173>    lea   0x2e30(%rip),%rax      # 0x555555558250 <chaff9>
0x555555555420 <main+180>    push  %rax
0x555555555421 <main+181>    lea   -0x2f(%rbp),%rax
>0x555555555425 <main+185>    push  %rax
0x555555555426 <main+186>    lea   0xc0f(%rip),%rax      # 0x55555555603c
0x55555555542d <main+193>    push  %rax
0x55555555542e <main+194>    lea   0xbec(%rip),%r9      # 0x555555556021
0x555555555435 <main+201>    lea   0xbef(%rip),%r8      # 0x55555555602a
0x55555555543c <main+208>    lea   0x2d9d(%rip),%rcx     # 0x5555555581e0 <chaff2>
0x555555555443 <main+215>    lea   0x2d86(%rip),%rdx     # 0x5555555581d0 <chaff1>
0x55555555544a <main+222>    lea   0x2d6f(%rip),%rsi     # 0x5555555581c0 <chaff0>
0x555555555451 <main+229>    lea   0xbdb(%rip),%rdi     # 0x555555556033

native process 3530 In: main
0x8000000000db0: <error: Cannot access memory at address 0x8000000000db0>
(gdb) si
0x000555555555419 in main ()
0x000555555555420 in main ()
(gdb) x/s $rbp+0x2e30
0x8000000000db0: <error: Cannot access memory at address 0x8000000000db0>
(gdb) x/s $rax
0x555555558250 <chaff9>:      "SeCmLYB0"
(gdb) si
0x000555555555421 in main ()
0x000555555555425 in main ()
(gdb) x/s $rax
0x7fffffffdf51: "MzRjZDEw"
(gdb) 
```