#### [Linear recurrence](http://timisoaractf.ro/challenge?id=23) (200pts) First to solve this challenge!

This problem gave us a linear recurrence defined by the first N terms ( 2 <= N <= 4 ). The fastest way I knew to solve this was to use matrix exponentiation. A great explanation of the technique can be found in the Competitive Programmer’s Handbook (<https://cses.fi/book/book.pdf>) at page 220. I recommend this book!

Anyway, the final script I used to solve this problem is as follows:

|  |
| --- |
| #!/usr/bin/env python2 # -\*- coding: utf-8 -\*- # This exploit template was generated via: # $ pwn template --host 89.38.208.143 --port 22022 from pwn import \*  # Set up pwntools for the correct architecture context.update(arch='i386') exe = './path/to/binary'  # Many built-in settings can be controlled on the command-line and show up # in "args". For example, to dump all data sent/received, and disable ASLR # for all created processes... # ./exploit.py DEBUG NOASLR # ./exploit.py GDB HOST=example.com PORT=4141 host = args.HOST or '89.38.208.143' port = int(args.PORT or 22022)  def local(argv=[], \*a, \*\*kw):  '''Execute the target binary locally'''  if args.GDB:  return gdb.debug([exe] + argv, gdbscript=gdbscript, \*a, \*\*kw)  else:  return process([exe] + argv, \*a, \*\*kw)  def remote(argv=[], \*a, \*\*kw):  '''Connect to the process on the remote host'''  io = connect(host, port)  if args.GDB:  gdb.attach(io, gdbscript=gdbscript)  return io  def start(argv=[], \*a, \*\*kw):  '''Start the exploit against the target.'''  if args.LOCAL:  return local(argv, \*a, \*\*kw)  else:  return remote(argv, \*a, \*\*kw)  # Specify your GDB script here for debugging # GDB will be launched if the exploit is run via e.g. # ./exploit.py GDB gdbscript = ''' continue '''.format(\*\*locals())  #=========================================================== # EXPLOIT GOES HERE #=========================================================== def multiply(a, b):  n = len(a) #assume ever mat as square nxn  c = [[0 for x in range(n)] for x in range(n)]  for i in range(n):  for j in range(n):  for k in range(n):  c[i][j] += (a[i][k] \* b[k][j]) % 666013  c[i][j] %= 666013  return c def lgput(a, b):  n = len(a)  res = [[0 for x in range(n)] for x in range(n)]  for i in range(n):  res[i][i] = 1   while(b > 1):  if(b % 2 == 0):  a = multiply(a, a)  else:  res = multiply(res, a)  a = multiply(a, a)  b /= 2  return multiply(res, a) io = start() for t in range(10):  io.recvuntil('/10:')   n = int(io.recvuntil(' ').strip(), 10)   #print n   k = int(io.recvuntil('\n'), 10)  #print k   initial\_d = []   coeffs = []   for i in range(n):  coeffs.append(int(io.recvuntil(' '), 10))  initial\_d.append(int(io.recvuntil(' '), 10))  #print coeffs, initial\_d    mat = [[0 for x in range(n)] for x in range(n)]   for i in range(n - 1):  mat[i][i + 1] = 1   for i in range(n):  mat[n - 1][i] = coeffs[n - 1 - i]    terms = [[0 for x in range(n)] for x in range(n)]   for i in range(n):  terms[i][0] = initial\_d[n - i - 1]  #print terms, mat  ans = multiply(lgput(mat, k - 1), terms)   print ans   io.sendline(str(ans[0][0]))  # shellcode = asm(shellcraft.sh()) # payload = fit({ # 32: 0xdeadbeef, # 'iaaa': [1, 2, 'Hello', 3] # }, length=128) # io.send(payload) # flag = io.recv(...) # log.success(flag)  io.interactive() |