hackme.inndy.tw的一些Writeup（5月30更新）

转载

baikeng3674 于 2017-10-21 21:35:00 发布 6128 收藏 1
原文链接：http://www.cnblogs.com/WangAoBo/p/hackme_inndy_writeup.html
版权

hackme.inndy.tw的一些Writeup（6月3日更新）

> 原文链接：http://www.cnblogs.com/WangAoBo/p/7706719.html

推荐一下https://hackme.inndy.tw/scoreboard/，上边有一些很好的针对新手的题目，但网上能搜到的Writeup很少，因此开了这篇博文记录一下部分目前解出的题目（主要是pwn和re），以后会跟着解题进度的推进逐步更新，同时遵循inndy师傅的规矩，只放思路，不放flag。

> ~~zwhubuntu师傅的部分题解：http://wxzwhubuntu.club:8088/index.php/2017/06/14/inndy/~~
>
> 聂师傅的部分题解: http://blog.csdn.net/niexinming
>
> 如果看了解题思路仍有问题，可以在我的github找到完整的脚本：pwn，reverse，欢迎star和follow

pwn

0x00 catflag

送分题，nc连接，cat flag即可

0x01 homework

根据题目提示，是数组越界的漏洞，第一次遇到这种漏洞觉得利用方式很巧妙。通过搜索字符串发现了get shell的函数**call_me_maybe**, **run_program**函数中定义了一个大小为10的数组，漏洞出现在**edit number**的选项里，当我们输入的**index**大于10时，程序是不会报错的，而是会继续朝着高地址**edit**数据，因此只需要**edit run_program**栈中的**ret**为**call_me_maybe**, 这样当我们退出**run_program**函数时，程序就会返回到**call_me_maybe**函数来**get_shell**，原理图如下：

> 有两点需要注意：

1. 因为edit number中修改的数据是通过**%d**输入的，因此需要以整形的方式输入**call_me_maybe**函数的地址
2. 修改ret的地址之后，还需要再输入一次0来退出**run_program**的循环来出发ret

payload:

0x02 ROP

刚看这道题的时候觉得无从下手，程序中既没有可以利用的函数，通过file命令查看是静态链接的也不能利用libc中的函数。后来注意到下一题提到了**ROPgadget**这个工具，才想到可以直接利用**ROPgadget**直接拼凑出ROP链，如下：

这样就只需要我们通过缓冲区溢出的漏洞返回到ROPgadget构造的ropchain就可以了

payload:

0x03 ROP2

本来根据提示以为这道题需要自己拼凑出ropchain，但后来发现这一题中存在**syscall**这个可以实现系统调用的函数，如**syscall(4, 1, &v4, 42)**即相当于**write(1, &v4, 42)**，**syscall(3, 0, &v1, 1024)**即相当于**read(0, &v1, 1024)**，syscall的第一个参数是系统函数的系统调用号，之后的参数依次为对应函数的参数，32位的系统调用号定义在**/usr/include/x86*64-linux-gnu/asm/unistd*32.h**中，可以看到execve的调用号为11，因此如果我们构造**syscall(11, "/bin/sh", 0, 0)**就相当于执行了**execve("/bin/sh", 0, 0)**即可get shell

> 介绍一个小trick，如下图，可以看出v4是提示字符串的开头，但因为IDA没有正确识别变量的类型，把提示字符串分成了v4~v15多个变量，v4的地址为bp-0x33，v15为bp-0x9，因此字符串长度为0x33 - 0x9 = 42
>
> 这时我们在v4上**y**一下，然后在弹出来的窗口上填入我们推断出来的数据类型**char v4[42]**，再点OK，这时IDA就能正确的识别出来v4的数据类型了，如下图：

payload1:

通过两次rop，实现get shell

payload2:

利用ROPgadget找到了一个pop4ret的gadget，利用pop4ret平衡堆栈，用一个ropchain实现getshell

需要注意，对于execve的第一个参数"/bin/sh"，我们可以用先写入一个固定地址（如bss段）的方式迂回传参。

0x04 toooomuch

放松心情的题，用IDA很容易可以得到通过验证的passcode，然后用 二分/XJB猜 猜对数字后，就可以拿到flag

0x05 toooomuch-2

题目已经提示利用缓冲区溢出通过shellcode来get shell，IDA查看程序流程，在gets函数中存在溢出

这样只要把通过ROP把shellcode写到bss段，再返回bss段执行shellcode即可

payload:

0x06 echo

看到echo，第一反应就是格式化字符串的题，分析文件后果然发现了很明显的格式化字符串漏洞

并且可以通过while循环多次利用，很经典的利用方式，先泄露出system_got中的system函数真实地址，再覆写printf_got为system_addr，之后通过fgets读入"/bin/sh"时，printf("/bin/sh")已经相当于system("/bin/sh")，即可get shell

payload:

> 需要注意如果leak的payload是**p32(system_got) + "%7$s"**的形式，那么泄露出的system_addr是从第4位到第8位（p32(system_got))占据了前4位，另外如果p32(system_got)中有\x00等bad char截断printf的话，可以调整payload形式为**%8$s + p32(system_got)**

0x07 echo2

很明显这一题也是格式化字符串的漏洞,与上一题的不同在于:

1. 64位,这意味着fmtstr_payload函数极有可能因为\x00不能用
2. 开启了PIE,也就是说got表等地址都是随机的,但因为elf中各个部分的偏移是固定的,因此可以通过泄露elf基址的方法来确定其他部分的真实地址

通过调试看出可以利用%41$p和%43$p泄露main地址与__libc_start_main的地址(0x23 + 6, 0x25 + 6):

泄露elf_base与libc_base的函数如下:

```
def getAddr():
    io.sendline("%41$p..%43$p..")
    elf_base = int(io.recvuntil("..", drop = True), 16) - 74 - 0x9b9#nm ./echo2
    libc_base = int(io.recvuntil("..", drop = True), 16) - 240 - libc_offset # this is for remote
    #  libc_base = int(io.recvuntil("..", drop = True), 16) - 241 - libc_offset #this is for local
    log.info("elf_base -> 0x%x" % elf_base)
    log.info("libc_base -> 0x%x" % libc_base)
    return elf_base + exit_got, libc_base + one_gadget
```

获得elf_base与libc_base基址后,按照正常的格式化字符串思路来就行,比如可以用one_gadget地址覆写exit函数的got表地址.

0x08 echo3

这道题目做出来后在和站主交流时发现使用了非预期解，不太容易说清楚，这阵比较忙就先只放exp，忙过了这一阵写一下对这道题的详细分析（估计要几个月后了），exp如下：

inndy_echo3

多说一句inndy人很nice

0x09 smash-the-stack

典型的canary leak，覆盖__libc_argv[0]为flag在内存中地址，触发__stack_chk_fail函数即可泄露flag

放一篇学习链接:http://veritas501.space/2017/04/28/%E8%AE%BAcanary%E7%9A%84%E5%87%A0%E7%A7%8D%E7%%8E

payload:

或者可以用更暴力的方法,不计算argv[0]到缓冲区的距离,用一个较大的值直接覆盖过去即可:

payload:

> 需要注意的是write函数的长度是由用户输入决定的，给buf一个较小的值即可

0x0A onepunch

> 本来以为onepunch的意思是构造好payload一发get shell， 后来才发现是一个字节一个字节的打

题中有一个任意地址写一个字节的漏洞，刚开始觉得无从下手，后来调试的时候发现代码段为rwxp权限，才觉得柳暗花明。

既然可以在代码段任意地址写，就意味着我们可以为所欲为的修改代码流程，因此就可以将

> .text:0000000000400767 jnz short loc_400773

修改为

> .text:0000000000400767 jnz 0x40071D

这样就构成了一个循环，接下来在合适的位置写shellcode，然后跳转到shellcode即可

payload:

> 至于怎么确定要patch的地址和值，我推荐keypatch，用keypatch修改完伪代码后，通过将 *Options -> General ->Number of Opcode byte*修改为非 0 值对比修改前后的字节码即可

0x0B rsbo

这道题的利用方法倒是第一次见到，对于read和write的第一个参数fd（文件描述符），fd = 0时代表标准输入stdin，1时代表标准输出stdout，2时代表标准错误stderr，**3~9则代表打开的文件**。这一题的利用方式利用方式就是利用rop先用open函数打开位于/home/rsbo/的flag，然后再用read(3,)把flag写到一个固定地址上，最后用write输出

payload:

0x0C leave_msg

经过分析代码的逻辑,绕过下边的限制,就可以覆写got表:

```
if ( v4 <= 64 && nptr != 45 )
  dword_804A060[v4] = (int)strdup(&buf);
```

而v4=atoi(&ntpr),经过搜索atoi函数会跳过字符串开头的空白字符,因此只要构造**(空格)-16**就可以同时绕过上边的两个限制来覆写puts的got表,然后再用\x00绕过strlen的限制就可以执行shellcode,payload如下:

此时的程序流程如下:

> 至于0x36是怎么来的,我是调试看出来的,如果哪位表哥有静态计算的方法,还请不吝赐教!

0x0D stack

这个题开了全保护，第一眼看上去挺吓人，但其实漏洞很容易发现，pop时并没有对下标作出检查，这就意味着我们可以通过一直pop利用数组越界从栈上leak，先通过调试看栈结构

可以看出，通过一直pop可以泄露**_IO_2_1_stdout_**的地址，进而确定libc的装载基址和one_gadget地址

```python
def getBase():
    #  debug()
    for i in xrange(15):
        io.sendlineafter("Cmd >>", "p")
        io.recvuntil("-> ")

    libc_base = (int(io.recvuntil("\n", drop = True)) & 0xffffffff)- libc.symbols["_IO_2_1_stdout_"]
    info("libc_base -> 0x%x" % libc_base)
    one_gadget = libc_base + one_gadget_offset

    return one_gadget
    #  return libc_base
```

通过**main+27, main+427**等地址可以泄露elf的装载基址（当然并没有用到），并且经过观察，**main+27, main+427分别是__x86_get_pc_thunk_dx和stack_push的返回地址**，这样我们只需要把main+427覆盖为one_gadget的地址，再次调用stack_push时，就可以返回到one_gadget，进而get shell，有一个坑点是0xf段的地址会发生数据溢出，需要我们转成int32的类型。

虽然听起来很麻烦但只要耐心调试并不难，这个题目的flag也说明了本题的重点就是leak from stack。

关于这个题有两点值得一提：

刚打开文件使用F5时报错

很容易搜索到这是因为函数的参数个数不匹配，我们定位到0x78D，发现540这个函数有一个参数，但IDA并没有正确识别

手动指定一个参数后（快捷键y指定类型），这个函数的报错就解决了

同样的方法修复之后的报错，就可以F5查看伪代码了

实现查看伪代码后，可以通过查看函数的具体实现进一步识别函数和修正函数参数

如上图的578函数，可以看出实际调用了hex(8128 + 56)这个地址，而这个地址在IDA中可以看出是scanf的地址，这样我们就可以通过scanf的参数列表进一步修正参数了，附一张我修复之后的图

这样在通过对比题目给出的源码，就很容易分析程序的功能了

这个题目开启了PIE保护，利用pwn.gdb.attach调试的时候和没有开启PIE保护的有些不同。

以前我调试开启了PIE保护elf的方式是Uriel师傅教我的先找elf装载基址

```python
from pwn import *
import sys, os
import re

wordSz = 4
hwordSz = 2
bits = 32
PIE = 0
mypid=0
context(arch='amd64', os='linux', log_level='debug')
  def leak(address, size):
     with open('/proc/%s/mem' % mypid) as mem:
        mem.seek(address)
        return mem.read(size)

  def findModuleBase(pid, mem):
     name = os.readlink('/proc/%s/exe' % pid)
     with open('/proc/%s/maps' % pid) as maps:
        for line in maps:
           if name in line:
              addr = int(line.split('-')[0], 16)
              mem.seek(addr)
              if mem.read(4) == "\x7fELF":
                 bitFormat = u8(leak(addr + 4, 1))
                 if bitFormat == 2:
                    global wordSz
                    global hwordSz
                    global bits
                    wordSz = 8
                    hwordSz = 4
                    bits = 64
                 return addr
     log.failure("Module's base address not found.")
     sys.exit(1)

  def debug(addr = 0):
      global mypid
      mypid = proc.pidof(r)[0]
      raw_input('debug:')
      with open('/proc/%s/mem' % mypid) as mem:
          moduleBase = findModuleBase(mypid, mem)
          gdb.attach(r, "set follow-fork-mode parent\nb *" + hex(moduleBase+addr))
```

View Code

但做这道题题时发现pwn.gdb.attach只有第一个参数是必须的

这样就可以先进入gdb，通过vmmap找到elf基址后在下断点进行调试了

0x0E very_overflow

这个题目给了源码，分析起来方便了不少。这个题的漏洞也很容易发现，虽然申请了长度为128*(sizeof(buffer))的缓冲区，但可以无限的add_note，这就意味着我们可以先重复add_note耗尽缓冲区，然后继续add_note和show_note时，就可以leak类似__libc_start_main这些信息来确定libc装载基址了，有了libc装载基址后，通过rop构造system("/bin/sh")或者one_gadget都可以求解

至于add_note多少次，通过调试可以很清楚的算出来

另外就是刚开始本地可以get shell，但远程连接很容易超时，后来把context.log_level换成了info，减少了打印花费的时间，又把io.sendlineafter换成了直接io.sendline，就不容易超时了

0x0F tictactoe-1

给的elf文件逆起来比较繁琐，通过反编译可以找到棋的源码，了解了程序的大体流程后，可以发现在落子时可以通过数组越界覆写GOT表

但因为程序有一个判负退出的功能，因此经过实验最多只能写三个字节，但对第一题而言，三个字节已经足够overwrite memset@got为打印flag的地址

第一题很快就解决了，至于tictatoe-2，虽然get shell拿到了flag，但根据flag形式需要用ret2dl_solve，等我用ret2dl_solve解决时再来补wp

reverse

0x00 helloworld:

0x01 simple:

都是水题，不再细说

0x03 pyyy

这一题用uncomplye6或者在线网站反编译后，得到的python代码都有大量的lambda操作，读起来十分费力，并且代码不能直接运行，仔细观察，代码中有一行

> this_is = 'Y-Combinator'

似乎是在暗示什么，google了一发，在 https://rosettacode.org/wiki/Y_combinator#Python 找到了Y-Combinator的介绍，即利用lambda等操作实现递归，于是按照网站上给出的python实例修改了代码，使其能够正确执行。再仔细观察，发现只要我们每轮的输入都和程序每轮计算的值相等即可得到flag，因此直接修改python代码，让输入等于程序计算好的值，修改好的代码如下：

```
1 # uncompyle6 version 2.12.0
2 # Python bytecode 2.7 (62211)
3 # Decompiled from: Python 2.7.13 (default, Jan 19 2017, 14:48:08)
4 # [GCC 6.3.0 20170118]
5 # Embedded file name: pyyy.py
6 # Compiled at: 2016-06-12 01:14:31
7 from fractions import gcd
```

```
 8 __import__('sys').setrecursionlimit(1048576)
 9 data =
'Tt1PJbKTTP+nCqHvVwojv9K8AmPWx1q1UCC7yAxMRIpddAlH+oIHgTET7KHS1SIZshfo2DOu8dUt6wORBvNVBpUSsuHa0S78KG+SCQtB2lr
4c1RPbMf0nR9SeSm1ptEY37y310SJMY28u6m4Y44qniGTi39ToHRTyxwsbHVuEjf480eeYAfSVvpWvS8Oy2bjvy0QMVEMSkyJ9p1QlGgyg3m
UnNCpSb96VgCaUe4aFu4YbOnOV3HUgYcgXs7IcCELyUeUci7mN8HSvNc93sST6mKl5SDryngxuURkmqLB3azioL6MLWZTg69j6dflQIhr8Rv
OLNwRURYRKa1g7CKkmhN4RytXn4nyK2UM/SoR+ntja1scBJTUo0I31x1wBJpT4HjDN47FLQWIkRW+2wnB3eEwO5+uSiQpzA8VaH7VGRrlU/B
FW4GqbaepzKPLdXQFBkNyBKzqzR/zA2GIrYbLIVScWJ19DqJCOyVLGeVIVXyzN1y327orYL2Ee3lRITnE3FouicRStaznIcw8xmxvukwVMRZ
IJ/vTu8Zc1WQIYEIFXMHozGuvzZgROZTyFihWNRCBBtoP9DJJALJb0pA1IKIb2zLh+pwGF40Y6y93D6weKejGPO+A0DBXH9vuLcCcCIvr/XP
QhO3jLKCBN+h9unuJKW3dyWxyaVPdR2V+BTw10VXolo7yaTH1GbR4TiVSB308mBOMwfchwihEe7RdMXvmXgaGarKkJe0NLUCd8jwhYII+Wym
jxO/xOz/ppOvNfAyIQksW0sggRPQTlgXSZ7MIVA1h66sGNljJ833MoFzWof3azLabaz1OrAJFqYXBg/myDsy1tV6rULSQ82hVR/TNnSmBGvy
EDJTrLSwHyj78NOrW4mUnlLGBnAgWfw6pW2lRK2jkNX9NM6DfLsRK8lwl85UP8CZSuNdcLmLwHTVMZGm/cNkZCtWRBlZqEggxGdIO44D+f4y
6ysnAk5/QzEwjIuecxEOb0jyV6dFui8g0c3Oxlhzcli0X8ToJFyeQRv1N9nokYZ07tFlG6m18kCToKz1qiH1U7kljXa6SvdORur5dWYLQ//g
whwppe7JlNda/cEoh92h96wRZDv1dSK/f1vz+mUeUyUlFY0iMjfw5eBXWZppNZi3ZtJcq5kllM2ACVFcxQWI3azM3ArOcqjosoiPjNoDYgKh
7w4k2Cd0kLYEHscz/njtJ1KEcwLtqs4nJ+gB2r4V9g03YgvY5E8JJtfJMKdaTedjtvEuif8FNlCK9DMnL1iLpWptJbdfO83Y7Y46XCqjZFBI
5o9Qtb78nLhMEM5/YTaNOM/wE/oJl5HI/i1X6kW3PKCsVubRkOkc2xawl6NYdLETjLvmrGhhI'
10 a =
1384297743827247992661626388675867697927484936093021404965338670080951734558799478947795963106395749747531924340527885231530345893644679683542515949630741511843376958857977216645433771365767283914419711631508678812306593568643923062435665604008133316579210134912828686127676127655726740161695877078021801849 07L
11 b =
1669733064888376163866575255608674720728926005823361708765820872597452046096219531271557043419866569983884763842689449916746221373215641690158922773946761118216257856605201248549491158480299929015700170034265160605875421515084578289933932692858111920619217778414140810240072465481761062708077557539592993474 99L
12 c =
1394069759046160109937810709689293869591377701617162762060093047881380644640038726008730921757941947422780657318360363196918209231108242974388738524314365520846825006789608158299139525042991219618516114863077708952684809726977768081087629989825196286733637273534178824366019144413853295760731981014167788206 19L
13 d =
1202478150402039718781564013360641958596174751092554889739831770905038410942700997980917509503103870209856314622417731948569282041763665652030993267115519508607269717294713310945910294762220363233013875849321697438583286531444277141338055882527520635201233492297817622692592906419029960304083898456084870180 53L
14 e =
1042679260526812323990220976935679455667921042663930429975924190845955908427925872898371621279723404023994832061791237208578933366585547347218588616325138151345580922637474230696634717430324850025242580530464799653861914221391155485264768362142750447769290646071689838317929951969737818499769050669678685137 07L
15 F = (a, b, c, d, e)
16 m =
880496167809374924436273771031704106620586070466893252755842415306105065093365785219582945259408317643302428678437340182291561691658281394125847173323301 1L
17 g = 67051725181167609293818569777421162357707866659797065037224862389521658445401L
18 z = []
19 for i, f in enumerate(F):
20     n = pow(f, m, g)
21     #https://rosettacode.org/wiki/Y_combinator#Python
22     this_is = 'Y-Combinator'
23     #  l = (lambda f: (lambda x: x(x))(lambda y: f(lambda *args: y(y)(*args))))
24     Y = lambda f: (lambda x: x(x))(lambda y: f(lambda *args: y(y)(*args)))
25     fun = lambda f: lambda x: (1 if x < 2 else f(x - 1) * x % n)
26     l = Y(fun)(g % 27777)
27 #   Y = lambda f: (lambda x: x(x))(lambda y: f(lambda *args: y(y)(*args)))
28 #   fac = lambda f: lambda n: (1 if n<2 else n*f(n-1))
29 #  >>> [ Y(fac)(i) for i in range(10) ]
30 #  [1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880]
31 #  >>> fib = lambda f: lambda n: 0 if n == 0 else (1 if n == 1 else f(n-1) + f(n-2))
32 #  >>> [ Y(fib)(i) for i in range(10) ]
33 #  [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
34     #  print l
35     #  c = raw_input('Channenge #%d:' % i)
36     c = l
37     if int(c) != 1:
```

```
37     if int(c) != 1:
38         print 'Wrong~'
39         exit()
40     z.append(l)
41
42 z.sort()
43 gg = '(flaSg\'7 \\h#GiQwt~66\x0csxCN]4sT{? Zx YCf6S>|~`\x0c$/}\'\r:4DjJFvm]
([sP%FMY"@=YS;CQ7T#zx42#$S_j0\\Lu^N31=r\x0b\t\tjVhhb_KM$|6]\nl!:V\rx8P[0m
;ho_\rR(0/~9HgE8!ec*AsGd[e|2&h!}GLGt\'=$\x0cbKFMnbez-q\\`I~];@$y#bj9K0xmI2#8
sl^gBNL@fUL\x0b\\9Ohf]c>Vj/>rnWXgLP#<+4$BG@,\'n a_7C:-}f(WO8Y\x0c2|(nTP!\'\\>^\'}-
7+AwBV!w7KUq4Qpg\tf.}Z7_!m+ypy=`3#\\=?9B4=?^}&\'~
Z@OH8\n0=6\x0b\tv\nl!G\'y4dQW5!~g~I*f"rz1{qQH{G9\x0c\'b\x0cp\x0bdu!2/\\@i4eG"If0A{-
)N=6GMC<U5/ds\rG&z>P1\nsq=5>dFZUWtjv\tX~^?9?Irwx\\5A!32N\x0bcVkx!f)sVY
Men\x0c\'ujN<"LJ\x0c5R4"\\\\XPVA\'m$~tj)Br}C}&kX2<|\np3XtaHB.P\'(E 4$dm!uDyC%u ["x[VYw=1aDJ (8V/a!J?
`_r:n7J88!a25AZ]#,ab?{%e\x0b]wN_}*Q:mh>@]u\t&6:Z*Fmr?U`cOHbAf7s@&5~L ,\tQ18 -Hg
q2nz%\x0ccUm=dz&h1(ozoZ)mrA=`HKo\n\'rXm}Z-l3]WgN\\NW<{o=)[V({7<N1.-
A8S"=;3sderb\tOZ$K\r0o/5\x0bMc76EGCWJ3IQpr7!QhbgzX8uGe3<w-g\'/j\'\tM4|9l?
i&tm_\n57X0B2rOpuB@H@%L_\r)&/q=LZa(%}""#if#Kq74xK?`jGFOn"8&^3Q-\r#]E$=!b^In0:$4VKPXP0UK=IK)Y\rstOT40=?
DyHor8j7O\\r/~ncJ5];cCT)c?OS0EM5m#V(-%"Tu:!UsE],0Dp  s@HErS]J{%oH54B&(zE.
(@5#2k\tJnNlnUEij\\.q/3HBpJNk*X(k5;DlqK\'\'fX\r}EBk_7\x0b:>8~\t+M@WJx.PO({/U}1}#TqjreG\nN{\rX>4EsJr0Pn\\Z\\a
L/-U<<{,Q;j\tF=7f\')+wH:p{G=_.s\\t-\x0bI\x0c*y\t1P:Y|/2xE<uo]~$>5k]FW+>fR<QA"
(Fj[LL(hzfQo#PJ;:*0kB~3]9uL[o.xue:VQ\t;9-Tu\tq|mzzhV_okP\t,d\rQ`]5Gf\x0c#gXB\x0cAH|)NI|K=KW-&p-
<b"3e.rO\x0cuK=\x0c^\r+MuLxCJ`UKaD\x0bBH&n+YVajZ(U7pwWtto3T10VLHwSJ\rK\t}\'F$l1:b2Bd\na=#t0iq}#!{1_)w$}
<Dp(borC\'\t?r6;,+k;a(Q3@B?RCWYEDrjZe!
[x=n_%S]rl{&fLr*mgCD;92/nNsaxKy/;\nr]sPK=`+YP>MmfB\n8O4/"}nE7r*=41f2\t37>K\'s$wpl;qS[`qzu\x0b\t\nuaU|b,C`4&
dRN~]7DnuTb2FhNHV!#Z2Hho\x0b[%.{O\t$q0\x0ch_@?
w@b8[I^{JL|O8]i8{p)A.w)14qK3JoyF%licZ~ga\rW[L:W\rtIvfWJjZUOvB\rS.Beav3!-@bw|PexJ
Pcw1\ry6!63B}]J])6fak/3r]W\tMeXt[uc(1_U lys{a1X\r%)[wwP3rhgNW{*d~_E%Q2htCt5ha@l0^0=\x0bwT\ni4/V;_\nM1rb?
w~Q)Dli4u\n`}1+D8"\t`@V~$9l$Uy**VnI (@Ga0<RxfmoNgJTtE-
aLH\rE5fMy7rk$)V\rL2Fv/AivOa"\nuX|70Xrw^D]%i%JyT\x0cc%cwZ/Wbp=IiY;/@nFEe>3=tM;K*`fReGoc5V/Ri?nXZ-
RW)\'\t<\x0cV>@X@-Ei4%sO%},B_pjc`s"@oKCmdgDhjUZT@?mb\'?
Q:F\x0bLJkPgjaFAc=rbrjAz$Zz\x0cq0GU!")xFOEF(x!3M\t:l83|}}HgGJJ#eT/I\x0b[|lK_n+;Wi/N^B4LzL.a(gVWq,zO6\'S|tb>R
X` ca*CO<w\x0ci =wc1,M~\x0bc`FYEs\r){+Ll8[I9-88m\t\\iK/\\hno-C[vX*3Hx:%:K\rt\x0cW!tj\'SOhqxP|k7cw Hm?I@?
P\'HmapG7$0#T(Auz]sjmd#\rFP/}53@-Kvmi(d%dZKLZ2LK\'e_E\x0bQmR 5/(irq4-EUyp<hB?[\tnU:p*xuzASM'
44
45 Y = lambda f: (lambda x: x(x))(lambda y: f(lambda *args: y(y)(*args)))
46 fun = lambda f: lambda n: (1 if n < 3 else f(n - 1) + f(n - 2))
47
48 print ''.join((gg[Y(fun)(i + 2)] for i in range(16))) % ''.join((data[pow((gcd(z[i % 5], z[(i + 1) % 5])
* 2 + 1) * g, F[i % 5] * (i * 2 + 1), len(data))] for i in range(32)))
49 #   Y = lambda f: (lambda x: x(x))(lambda y: f(lambda *args: y(y)(*args)))
50 #   fac = lambda f: lambda n: (1 if n<2 else n*f(n-1))
51 #  >>> [ Y(fac)(i) for i in range(10) ]
52 #  [1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880]
53 #  >>> fib = lambda f: lambda n: 0 if n == 0 else (1 if n == 1 else f(n-1) + f(n-2))
54 #  >>> [ Y(fib)(i) for i in range(10) ]
55 #  [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
56 #  okay decompiling pyyy.pyc
```

运行即可

0x04 accumulator

这个题目中check函数的F5代码有点混乱，单看伪代码很难分析，但通过调试可以快速确定check函数的整体流程是对输入的每一位求和，然后与0x601080这个数组进行比较，在这个过程中更新0x6013C0和0x6013B0两个全局变量用于寻址，程序的整体流程是先check(sha512(input))，然后check(input)，因为sha512是哈希函数，因此放弃了逆向求解。仔细分析，在check(input)时，使用的也是对input每一位求和，然后与数组比较的方法，而与第一次check(sha512(input))相比仅仅是两个用于寻址的全局变量不同，这样我们只要对数组逐位做差即可

```
 1 #!/usr/bin/env python
 2 # -*- coding: utf-8 -*-
 3 __Auther__ = 'M4x'
 4 
 5 data = [195, 255, 493, 584, 799, 929, 946, 1086, 1180, 1184, 1421, 1595, 1805, 1846, 2081, 2320, 2430, 2605, 2727, 2972, 3213, 3403, 3418, 3649, 3712, 3950, 3989, 4193, 4228, 4394, 4523, 4624, 4706, 4935, 4999, 5072, 5106, 5291, 5510, 5536, 5644, 5751, 5993, 6118, 6126, 6198, 6211, 6410, 6469, 6609, 6647, 6752, 6978, 7010, 7053, 7106, 7274, 7468, 7563, 7673, 7706, 7956, 8146, 8187, 8257, 8333, 8398, 8469, 8592, 8640, 8693, 8742, 8793, 8844, 8901, 8953, 9007, 9062, 9113, 9161, 9215, 9317, 9374, 9429, 9483, 9540, 9591, 9644, 9692, 9741, 9792, 9846, 9944, 9996, 10045, 10144, 10195, 10246, 10294, 10350, 10402, 10450, 10551, 10652, 10750, 10849, 10946, 11045, 11096, 11147, 11202, 11304, 11353, 11451, 11507, 11605, 11653, 11753, 11852, 11900, 11951, 12052, 12105, 12161, 12259, 12360, 12409, 12461, 12563, 12664, 12718, 12775, 12823, 12921, 12970, 13020, 13071, 13173, 13227, 13276, 13374, 13422, 13521, 13569, 13667, 13718, 13771, 13873, 13972, 14029, 14080, 14179, 14278, 14377, 14432, 14482, 14531, 14579, 14627, 14679, 14732, 14789, 14840, 14894, 14951, 15052, 15154, 15210, 15263, 15314, 15363, 15460, 15509, 15610, 15666, 15763, 15818, 15916, 15968, 16018, 16075, 16132, 16233, 16288, 16386, 16443, 16543, 16600, 16655, 16703, 16801, 16858, 16955, 17005, 17056, 17153, 17250, 17375]
 6 
 7 flag = ""
 8 for i in xrange(1, len(data)):
 9     flag += chr(data[i] - data[i - 1])
10 
11 print flag
```

View Code

> 再介绍一个IDA使用的trick，在提取数组时，先在数组的第一位上点d，使数组的类型转换成dd，然后再点a，使IDA把这一段连续的数据转成数组，然后可以使用alt+L快速选中数组范围（alt+L，鼠标滚轮，afl+L），然后通过shift+E就可以快速导出我们想要选择的数据了

0x05 gccc

该题的逻辑很简单,只有一个输入,下载文件后发现gccc.exe为.Net(C#)架构,因此用C#的反编译工具(这里用了)反编译得到题目代码如下:

```
// GrayCCC
public static void Main()
{
    Console.Write("Input the key: ");
    uint num;
    if (!uint.TryParse(Console.ReadLine().Trim(), out num))
    {
        Console.WriteLine("Invalid key");
        return;
    }
    string text = "";
    string text2 = "ABCDEFGHIJKLMNOPQRSTUVWXYZ{} ";
    int num2 = 0;
    byte[] array = new byte[]
    {
        164, 25, 4, 130, 126, 158, 91, 199, 173, 252, 239, 143, 150, 251, 126,
        39, 104, 104, 146, 208, 249, 9, 219, 208, 101, 182, 62, 92, 6, 27, 5, 46
    };
    byte b = 0;
    while (num != 0u)
    {
        char c = (char)(array[num2] ^ (byte)num ^ b);
        if (!text2.Contains(new string(c, 1)))
        {
            Console.WriteLine("Invalid key");
            return;
        }
        text += c;
        b ^= array[num2++];
        num >>= 1;
    }
    if (text.Substring(0, 5) != "FLAG{" || text.Substring(31, 1) != "}")
    {
        Console.WriteLine("Invalid key");
        return;
    }
    Console.WriteLine("Your flag is: " + text);
}
```

View Code

可以看到,逻辑很简单,对输入进行了32轮验证,通过每一层验证即可得到flag.

但逻辑简单并不意味着容易解决,仔细分析代码可以看出解题的关键在求出num的值,num经过32次右移一位后等于0,因此取值范围为**[2 ^ 31, 2 ^ 32)**, 在该范围内找到满足32轮验证的值即可,但在找到该值时遇到了问题,如果单纯爆破的话,2 ^ 32 - 2 ^ 31次爆破远远超出了能接受的范围,因此考虑用z3来解决这个多约束问题.

该题的约束条件有如下几个:

- 0~5位为"FLAG{"
- 最后一位(31)为"}"
- 6~30位需在"ABCDEFGHIJKLMNOPQRSTUVWXYZ{} "内

因此可以写出z3的代码如下:

⊞ ⊟

```
def getNum():
    b = 0
    num2 = 0
    # 2 ** 30 <= num < 2 ** 31
    s = Solver()
    num = BitVec('num', 64)
    s.add(num >= 2 ** 31)
    s.add(num < 2 ** 32)
    # s.add(num > 1510650850)

    for i in xrange(32):
        if i < 5:
            s.add(((array[num2] ^ (num & 0x7f) ^ b) & 0x7f) == ord('FLAG{'[i]))
        elif 5 <= i < 31:
            s.add(
                    Or(
                        And(
                            ((array[num2] ^ (num & 0x7f) ^ b) & 0x7f) >= 65,
                            ((array[num2] ^ (num & 0x7f) ^ b) & 0x7f) <= 90,
                        ),

                        # ((array[num2] ^ (num & 0x7f) ^ b) & 0x7f) == ord('{'),
                        # ((array[num2] ^ (num & 0x7f) ^ b) & 0x7f) == ord('}'),
                        ((array[num2] ^ (num & 0x7f) ^ b) & 0x7f) == ord(' ')

                    )
                )
                elif i == 31:
                        s.add(((array[num2] ^ (num & 0x7f) ^ b) & 0x7f) == ord('}'))
        b ^= array[num2]
        b &= 0x7f
        num2 += 1
        num >>= 1

    if s.check() == sat:
        print s.model()
                #bug
        #  print s.model()[num].as_long()
    # while s.check() == sat:
        # print s.model()[num]
        # s.add(Or(num != s.model()[num].as_long()))
```

View Code

运行,num的值就秒出了

> 注释掉的部分是一个还没解决的bug,本来是想通过添加约束跑出所有的可行解,但可能是因为用了BitVec导致在数据类型的转换上有些问题,现在只跑出了一组解.等到考完试再来修这个bug

有了num的值,接下来有很简单了,可用python求解

```
#!/usr/bin/env python
```

```python
# -*- coding: utf-8 -*-
__Auther__ = "M4x"

import pdb
from z3 import *
array = [164,25, 4, 130, 126, 158, 91, 199, 173, 252, 239, 143, 150,
251, 126, 39, 104, 104, 146, 208, 249, 9, 219, 208, 101,
182, 62, 92, 6, 27, 5, 46]
# print len(array)

def getNum():
    b = 0
    num2 = 0
    # 2 ** 30 <= num < 2 ** 31
    s = Solver()
    num = BitVec('num', 64)
    s.add(num >= 2 ** 31)
    s.add(num < 2 ** 32)
    # s.add(num > 1510650850)

    for i in xrange(32):
        if i < 5:
            s.add(((array[num2] ^ (num & 0x7f) ^ b) & 0x7f) == ord('FLAG{'[i]))
        elif 5 <= i < 31:
            s.add(
                    Or(
                        And(
                            ((array[num2] ^ (num & 0x7f) ^ b) & 0x7f) >= 65,
                            ((array[num2] ^ (num & 0x7f) ^ b) & 0x7f) <= 90,
                        ),

                        # ((array[num2] ^ (num & 0x7f) ^ b) & 0x7f) == ord('{'),
                        # ((array[num2] ^ (num & 0x7f) ^ b) & 0x7f) == ord('}'),
                        ((array[num2] ^ (num & 0x7f) ^ b) & 0x7f) == ord(' ')

                    )
                )
                elif i == 31:
                        s.add(((array[num2] ^ (num & 0x7f) ^ b) & 0x7f) == ord('}'))
        b ^= array[num2]
        b &= 0x7f
        num2 += 1
        num >>= 1

    if s.check() == sat:
        print s.model()
                #bug
        #  print s.model()[num].as_long()
    # while s.check() == sat:
        # print s.model()[num]
        # s.add(Or(num != s.model()[num].as_long()))

def getFlag():
    text2 = "ABCDEFGHIJKLMNOPQRSTUVWXYZ{} "
    num = 3658134498
    num2 = 0
    b = 0
    flag = ""

    while num:
```

```
        c = chr((array[num2] ^ (num & 0x7f) ^ b) & 0x7f)
        if c not in text2:
            print ord(c)
        flag += c
        b ^= array[num2]
        num2 += 1
        num >>= 1
        print flag

#  getNum()
getFlag()
```

View Code

也可简单的修改反编译得到的c#代码,运行C#得到flag(推荐该方法)

⊞ ⊟

```
// GrayCCC
using System;
namespace GrayCCC
{
    class GCCC
    {
        public static void Main()
        {
            // Console.Write("Input the key: ");
            // uint num;
            // if (!uint.TryParse(Console.ReadLine().Trim(), out num))
            // {
            //     Console.WriteLine("Invalid key");
            //     return;
            // }
            string text = "";
            string text2 = "ABCDEFGHIJKLMNOPQRSTUVWXYZ{} ";
            int num2 = 0;
            uint num = 3658134498;
            byte[] array = new byte[]
            {
                164,25, 4, 130, 126, 158, 91, 199, 173, 252, 239, 143, 150,
                251, 126, 39, 104, 104, 146, 208, 249, 9, 219, 208, 101,
                182, 62, 92, 6, 27, 5, 46
            };

            byte b = 0;
            while (num != 0u)
            {
                char c = (char)(array[num2] ^ (byte)num ^ b);
                if (!text2.Contains(new string(c, 1)))
                {
                    Console.WriteLine("Invalid key");
                    return;
                }
                text += c;
                b ^= array[num2++];
                num >>= 1;
                Console.WriteLine(text);
            }
            if (text.Substring(0, 5) != "FLAG{" || text.Substring(31, 1) != "}")
            {
                Console.WriteLine("Invalid key");
                return;
            }
            Console.WriteLine("Your flag is: " + text);
        }
    }
}
```

View Code

> 可以看出,通过z3求解此类多约束问题能极大地提高效率.
>
> 另外听Kira师傅说这道题也可以逐位爆破,也等到考完试再来填坑吧.

0x06 ccc

IDA F5后程序的流程很清楚，唯一一处有困惑的是不知道下图中的crc32函数是按照什么规则加密的并且返回值是什么

根据for循环中的i = 3， i += 3等信息可以推测v3是每3位计算一次，但究竟是a2[0: 3], a2[3: 6]这种方式还是a2[0: 3], a2[0: 6]这种方式还不得而知，静态分析crc32函数有很麻烦。这时，就可以用gdb动态调试来验证加密的方式。

首先在伪代码页右键，Copy to Assembly，方便我们在汇编页快速定位，如下定位到关键地址0x8048558

直接在gdb中下断点，r运行到断点，输入42位字符，此时程序运行到了断点处

n，执行下一步，此时crc32的返回值即保存在了eax寄存器中（函数调用约定）

可以发现，第一次调用crc32函数时，函数对输入的前3位做了计算，接下来调用的计算方式可以类比动态调试得到，最终发现crc32是按照a[0: 3], a[0: 6], a[0: 9]这样的方式进行计算的，每次循环计算的结果再与hashes中的只比较即可

> 再补充一个trick，这个操作是跟w1tcher表哥学到的
>
> 我们知道crc32计算的结果是8位16进制数，而IDA默认以两位16进制数显示数据，这是我们可以在第一个数据上右键，data，使前四个转成8位16进制数形式（或者快捷键D），然后再右键，array，点击OK，即可默认把下边的数据按第一组的形式展现出来

每次爆破3位即可，脚本如下：

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
__Auther__ = 'M4x'
import binascii
import string
import pdb

hashes = [0x0D641596F, 0x80A3E990, 0x0C98D5C9B, 0x0D05AFAF, 0x1372A12D, 0x5D5F117B, 0x4001FBFD, 0x0A7D2D56B,
0x7D04FB7E, 0x2E42895E, 0x61C97EB3, 0x84AB43C3, 0x9FC129DD, 0x0F4592F4D]

def crc(s, cnt):
    for a in dic:
        for b in dic:
            for c in dic:
                ans = s + a + b + c
                #  print tmp
                if (binascii.crc32(ans) & 0xffffffff) == hashes[cnt]:
                    #  pdb.set_trace()
                    return ans

#  dic = string.ascii_letters + string.digits + " _-{},"
dic = string.printable
#  ans = 'FLAG{'
ans = ''
cnt = 0
while True:
    try:
        ans = crc(ans, cnt)
        cnt += 1
        print ans
    except:
        break
```

View Code

0x07 bitx

IDA中可以看到，输入是与0x804A040地址上的一组数据进行比较的（IDA伪代码页，右键，Hide casts可以隐藏数据类型，看起来美观不少），程序逻辑很简单，直接放脚本

提取数据可以直接粘贴复制，也可以直接用idc或者ida python等脚本，如下的idc脚本即可dump出内存中的数据

解题脚本如下

⊞ ⊟

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
__Auther__ = 'M4x'

data = [0x8f, 0xaa, 0x85, 0xa0, 0x48, 0xac, 0x40, 0x95, 0xb6, 0x16, 0xbe, 0x40, 0xb4, 0x16, 0x97, 0xb1,
0xbe, 0xbc, 0x16, 0xb1, 0xbc, 0x16, 0x9d, 0x95, 0xbc, 0x41, 0x16, 0x36, 0x42, 0x95, 0x95, 0x16, 0x40, 0xb1,
0xbe, 0xb2, 0x16, 0x36, 0x42, 0x3d, 0x3d, 0x49, 0x00]

ans = ""
for i in xrange(42):
    ans += chr((((data[i] & 0xAA) >> 1) | (2 * (data[i] & 0x55))) - 9)

print ans
```

View Code

0x08 2018-rev

这个我也不知道用的是不是预期解。

直接运行文件，提示

> argc == 2018 && argv[0][0] == 1 && envp[0][0] == 1

经过调试写了一个gdb脚本绕过第一层验证

```
b *0x4005F0
r

#set args
set $rdi=2018
#dumpargs --force
set *((long *)0x7fffffffe332)=0x0101010101010101
set *((long *)0x7fffffffe34d)=0x0101010101010101
c

# zdump /etc/localtime
# 因为ASLR请自行修改对应地址
```

第二层验证提示

> Bad timing, you should open this at 2018/1/1 00:00:00 (UTC) :(

刚开始的想法也是和第一层验证一样设置运行过程中各个参数的值，但后来发现这些很难找，卡了一段时间后如梦初醒，程序的运行时间是从**/etc/localtime**获取的(代码中可以看出)，只要保证程序运行的瞬间**zdump /etc/localtime**的值为**2018/1/1 00:00:00 (UTC)**即可，关于如何设置时间，刚开始的想法是在gdb调试过程中set，尝试了很久，最后还是写了一个shellscript

```bash
#!/usr/bin/env bash

while :
do
    sudo date -us "2018-01-01 00:00:00"
done
```

这样一直运行setTime.sh，然后gdb导入gdb脚本，即可获得flag

这个题好像是文件放错了，最后的flag不完全正确，但很容易能猜出正确的flag

0x09 what-the-hell

这个题有点意思,看题目的提示好像是要优化算法,但我用了非预期解.我把思路和写残了的代码放出来,供大家参考.

用IDA打开,calc_key3的逻辑很清晰,就是运算比较麻烦根据题意应该就是要优化这里了.但看到这里的条件时我忽然想到了前一阵看到的z3,试了一下,很快就跑出来了,计算a1,a2的z3代码如下:

```python
def getInput():
    a2 = BitVec('a2', 32)
    a1 = BitVec('a1', 32)
    #  print type(a2)
    #  print type(a1)

    s = Solver()
    s.add(a2 * a1 == 0xDDC34132)
    s.add(((a1 - a2) & 0xFFF) == 0xCDF)
    s.add((a1 ^ 0x7E) * (a2 + 16) == 0x732092BE)

    while s.check() == sat:
        if isPrime(s.model()[a1].as_long()):
            print s.model()
        s.add(Or(a1 != s.model()[a1], a2 != s.model()[a2]))
    else:
        print "unset"
```

跑出了符合题意的两组解:

可以看出,what函数实际上是一个求斐波那契数列的函数

但这里有一个坑点,result是int型的,要考虑溢出,经过测试,当a1=4284256177时,函数有解,返回值1095061718431,这样我们就得到了decrypt_flag的三个参数,就可以写脚本跑出flag了,脚本如下:

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
__Auther__ = 'M4x'

from z3 import *
```

```python
from z3 import *
from libnum import prime_test as isPrime
from libnum import n2s

def getInput():
    a2 = BitVec('a2', 32)
    a1 = BitVec('a1', 32)
    #  print type(a2)
    #  print type(a1)

    s = Solver()
    s.add(a2 * a1 == 0xDDC34132)
    s.add(((a1 - a2) & 0xFFF) == 0xCDF)
    s.add((a1 ^ 0x7E) * (a2 + 16) == 0x732092BE)

    while s.check() == sat:
        if isPrime(s.model()[a1].as_long()):
            print s.model()
        s.add(Or(a1 != s.model()[a1], a2 != s.model()[a2]))
    else:
        print "unset"

def getKey3():
    fib = [0, 1]
    for i in xrange(2, 9999999):
        #  print i - 1, (fib[i - 1] & 0xffffffff)
        fib.append(fib[i - 1] + fib[i - 2])
        if (fib[i] & 0xffffffff) == 4284256177:
            return i * 1234567890 + 1
def getFlag():
    junk_data = [
  0x09, 0x23, 0x8C, 0xB9, 0x2F, 0x19, 0x8D, 0xF8, 0xF3, 0x79,   0x81, 0x87, 0x93, 0x99, 0x35, 0x52, 0x9C,
0xF0, 0x34, 0x99,   0x23, 0xB1, 0x84, 0x1D, 0xF0, 0x8F, 0x7E, 0x45, 0x0F, 0xCB,   0x40, 0xF8, 0x4E, 0xD1,
0x42, 0x29, 0x76, 0x17, 0x43, 0xE1,   0xAC, 0x04, 0x37, 0xA0, 0xE4, 0x30, 0x59, 0xA9, 0x68, 0xD9,   0x1C,
0x96, 0xFC, 0x1D, 0x85, 0xEA, 0xD2, 0x94, 0x07, 0x90,   0x09, 0xD2, 0xC9, 0x19, 0x86, 0xC9, 0xDC, 0x24,
0x6F, 0x3B,   0x5C, 0x92, 0x4C, 0x9F, 0xD9, 0x50, 0xDD, 0x98, 0x37, 0x1C,   0xB1, 0xDA, 0xA5, 0x44, 0xF2,
0x8E, 0x43, 0x66, 0x91, 0xA3,   0xDF, 0xAF, 0x3A, 0x7E, 0x65, 0x91, 0x19, 0x22, 0xFD, 0xFE,   0x14, 0xBA,
0x0A, 0xE1, 0xB9, 0x61, 0x73, 0x86, 0xE1, 0x96,   0xC1, 0x67, 0xCE, 0x06, 0x25, 0x74, 0xF0, 0x2E, 0xA3,
0xBB,   0xED, 0x68, 0x3E, 0x53, 0x30, 0x43, 0x0E, 0x53, 0xB8, 0x8A,   0x9C, 0x95, 0x41, 0xC3, 0xB0, 0x25,
0x1C, 0xCB, 0x38, 0x86,   0xA6, 0x7A, 0x6F, 0xF2, 0x63, 0x0A, 0x19, 0x7C, 0x07, 0xDA,   0x6F, 0xA2, 0x4E,
0xD2, 0x74, 0x4A, 0xF9, 0xAF, 0xC2, 0x9C,   0xFD, 0x89, 0xE6, 0x04, 0x11, 0xF6, 0x6F, 0xF5, 0x98, 0x55,
0x9D, 0x37, 0x12, 0xF2, 0xA6, 0x66, 0xBE, 0x85, 0x87, 0x8E,   0x87, 0x64, 0x5E, 0xA0, 0x61, 0x52, 0xD8,
0xBB, 0x39, 0x3D,   0x7B, 0xD2, 0x47, 0x27, 0x37, 0x30, 0xB5, 0xF8, 0x90, 0xFC,   0x50, 0xF3, 0xC1, 0x5C,
0x6B, 0xA4, 0xBE, 0x8D, 0xA5, 0xEA,   0xDD, 0x72, 0xF2, 0x28, 0xE1, 0x74, 0xEF, 0x07, 0x10, 0xCF,   0x39,
0x7D, 0x58, 0xE7, 0x46, 0x09, 0x04, 0xE9, 0xE9, 0x37,   0xD7, 0xE1, 0x20, 0xF9, 0xC2, 0x54, 0x28, 0xE7,
0x30, 0xE8,   0x86, 0x58, 0x77, 0x6C, 0x7D, 0x2E, 0x00, 0xCE, 0xCC, 0x9C,   0xFB, 0xA3, 0x8D, 0xD1, 0x04,
0x98, 0x9D, 0x4F, 0xE8, 0x1F,   0x60, 0x3A, 0x8A, 0x5B, 0x1A, 0x11, 0x55, 0xF0, 0x6B, 0xCF,   0xD8, 0x6D,
0x75, 0x30, 0x9A, 0xD8, 0xD8, 0x5D, 0x2E, 0x90,   0x7E, 0x43, 0x5C, 0xEB, 0x3F, 0x26, 0x78, 0xAF, 0xB3,
0xB0,   0xC3, 0x1C, 0xE9, 0xAB, 0x94, 0xE6, 0xC1, 0x49, 0x25, 0x4B,   0xAA, 0xFF, 0x59, 0xE1, 0x11, 0x48,
0x3C, 0xB9, 0x16, 0x67,   0x27, 0xF9, 0xA0, 0x29, 0x68, 0x2E, 0xFB, 0x45, 0x5D, 0x29,   0x12, 0x0A, 0x36,
0x04, 0x54, 0xB3, 0xCF, 0x87, 0x24, 0x37,   0x8E, 0x7C, 0x5A, 0xEF, 0xF8, 0x33, 0xE2, 0xE0, 0x89, 0x83,
0xA8, 0x4D, 0x72, 0x28, 0x80, 0xAA, 0xD4, 0x0E, 0xDD, 0x72,   0xA5, 0x0B, 0xAD, 0x85, 0x6F, 0xEE, 0x44,
0xAD, 0x43, 0x7D,   0x30, 0xC2, 0x15, 0xC9, 0x72, 0x12, 0x53, 0x8A, 0x37, 0x9D,   0xF2, 0x64, 0x1D, 0x21,
0x5E, 0x49, 0x78, 0x54, 0xC0, 0xF0,   0xA9, 0x81, 0xE3, 0x32, 0xD4, 0x99, 0x81, 0x88, 0x64, 0xFE,   0x20,
0x92, 0x89, 0xD0, 0xC9, 0x5A, 0xCE, 0xFA, 0xB5, 0xE4,   0x2A, 0x9D, 0x50, 0xAB, 0x32, 0x35, 0x8D, 0x31,
0x4C, 0x94,   0x6C, 0xC0, 0xEF, 0xF4, 0xE2, 0x40, 0xF7, 0x47, 0x51, 0xDB,   0x1C, 0x6D, 0x3B, 0x6B, 0xEA,
0xDA, 0x16, 0x9A, 0x27, 0x68,   0xA3, 0x73, 0xBF, 0x9D, 0x40, 0x8F, 0x07, 0xF3, 0xC7, 0x65,   0x57, 0xB7,
0x7E, 0x0C, 0xEA, 0xC9, 0x9F, 0x7F, 0x46, 0x82,   0xE6, 0x5C, 0xE6, 0xDF, 0xFE, 0x42, 0x41, 0x12, 0x62,
0x33,   0x74, 0xFF, 0xE9, 0x52, 0xD1, 0x0F, 0x75, 0x88, 0x43, 0x17,   0x02, 0x5A, 0x9E, 0x29, 0xAD, 0x40,
0x62, 0xDB, 0x1F, 0x2C,   0xE7, 0xA8, 0x6E, 0xAC, 0x62, 0xC4, 0xBE, 0xEC, 0x98, 0xB3,   0xE9, 0x44, 0xD4,
```

0x3E, 0xC3, 0x9E, 0x0F, 0xBC, 0xAD, 0xC6, 0x28, 0x28, 0x95, 0x93, 0xD1, 0xD5, 0x03, 0xAA, 0x78, 0xE3,
0x0D, 0x20, 0x90, 0x58, 0x82, 0xCC, 0x5F, 0x46, 0xF9, 0x2C, 0x17, 0x55, 0xDB, 0x96, 0x0A, 0x34, 0x69,
0x6B, 0x87, 0x2B, 0xB2, 0x45, 0x9D, 0x7C, 0xEA, 0xF3, 0xAB, 0x19, 0x6A, 0xE3, 0x73, 0x9D, 0x84, 0x6C,
0x3A, 0x04, 0xB5, 0x07, 0x3D, 0x10, 0x3E, 0x67, 0x5E, 0x53, 0x86, 0xDB, 0xA0, 0x39, 0xAB, 0xE0, 0x06,
0x22, 0x27, 0x8C, 0x81, 0xD3, 0xC6, 0x1F, 0x15, 0x35, 0x8D, 0x26, 0x8D, 0x67, 0x03, 0xBD, 0xC3, 0x76,
0xF7, 0x27, 0x29, 0x82, 0xAF, 0x64, 0x9D, 0x15, 0x0F, 0xBE, 0xAD, 0xAB, 0x39, 0x50, 0xD7, 0xB9, 0x1B,
0x0A, 0x3F, 0x99, 0xCC, 0x6A, 0xF5, 0xFF, 0x5B, 0xDE, 0x9F, 0xD1, 0x4D, 0xFC, 0xF4, 0x21, 0x83, 0xD0,
0x33, 0xAB, 0xA4, 0x3E, 0x3B, 0x3A, 0x67, 0x41, 0xDE, 0x93, 0xB1, 0x2A, 0xC4, 0x98, 0xDA, 0xEA, 0x51,
0xFD, 0xCF, 0xEB, 0xF0, 0xC7, 0x1F, 0xF6, 0x4C, 0xFC, 0x04, 0xEF, 0x24, 0x51, 0xBF, 0x0D, 0xB0, 0x50,
0x2C, 0x06, 0xAA, 0x0E, 0x2F, 0x74, 0xB0, 0xD5, 0x27, 0xE7, 0xD3, 0xC4, 0xA7, 0x57, 0x0D, 0x31, 0xCE,
0xD2, 0x5F, 0x6F, 0x99, 0x43, 0xDB, 0x93, 0x59, 0x24, 0xC0, 0xA8, 0x29, 0x3F, 0xDA, 0x70, 0xBD, 0x92,
0x8A, 0xC5, 0x76, 0xF9, 0x31, 0x8B, 0x98, 0xD6, 0x0A, 0x7C, 0xA0, 0x8D, 0x9B, 0x96, 0x4A, 0x77, 0x5B,
0xC2, 0xE5, 0x46, 0x72, 0x28, 0x4F, 0x54, 0x44, 0x06, 0xB4, 0xE5, 0xB2, 0x6C, 0xEF, 0x4E, 0x1E, 0x7E,
0xAE, 0x0A, 0xC0, 0x7D, 0x1E, 0x6E, 0x80, 0x3A, 0xDF, 0x88, 0x07, 0x4B, 0xF8, 0xCE, 0x3A, 0x40, 0x60,
0x6F, 0xDA, 0x9F, 0xE4, 0xD9, 0x58, 0xA3, 0x19, 0xEC, 0x5A, 0xD8, 0x85, 0x52, 0x1E, 0xA8, 0xCA, 0x04,
0xDC, 0x5D, 0xD2, 0x77, 0x45, 0x35, 0xB0, 0x5A, 0xD1, 0xCD, 0xDC, 0x30, 0xA6, 0x14, 0xA6, 0xA1, 0xBF,
0x24, 0xE1, 0xDE, 0xE6, 0xEF, 0xA9, 0x0E, 0x00, 0x64, 0x5D, 0xEF, 0x11, 0x4A, 0xF3, 0x38, 0x52, 0x86,
0x81, 0x6F, 0x42, 0xFB, 0x8B, 0x4B, 0x36, 0xFB, 0x79, 0x9D, 0x82, 0xBC, 0x0D, 0x01, 0x14, 0x42, 0x86,
0xD7, 0x65, 0xB4, 0x51, 0xBF, 0xEC, 0x64, 0xE4, 0x61, 0x21, 0x63, 0x99, 0xD3, 0xC5, 0xFE, 0x58, 0x0A,
0xF5, 0xA1, 0xD5, 0xB0, 0xD9, 0xB4, 0x8A, 0x02, 0xC7, 0x50, 0xDE, 0xDE, 0xF2, 0xBE, 0x13, 0xF8, 0x3F,
0x23, 0x51, 0x4C, 0x19, 0x40, 0x74, 0xA6, 0x35, 0xBA, 0x4B, 0x71, 0x1B, 0xAE, 0xFE, 0x43, 0x8F, 0xA4,
0x25, 0xA5, 0xE5, 0x31, 0xB3, 0x17, 0x00, 0x83, 0x34, 0x4A, 0xBA, 0x05, 0xCF, 0xBB, 0xB8, 0x67, 0x25,
0xE0, 0xD3, 0x53, 0xFC, 0xAA, 0xBA, 0xB3, 0x6C, 0x8A, 0xEC, 0x8F, 0x9C, 0xDB, 0x47, 0x05, 0x8E, 0x5A,
0x3E, 0xD4, 0x7B, 0x5F, 0xC5, 0x42, 0xD1, 0x6C, 0x2C, 0x99, 0xBA, 0xFD, 0x9D, 0x6B, 0x52, 0xD2, 0x34,
0x86, 0x6A, 0x5D, 0x5E, 0x50, 0xB3, 0x58, 0xD4, 0x3A, 0xB7, 0x12, 0x46, 0x0E, 0x40, 0x81, 0xA5, 0x21,
0x5D, 0x5E, 0x63, 0xE5, 0x3B, 0x30, 0x3B, 0x6E, 0x13, 0x73, 0x36, 0x20, 0x3C, 0xE3, 0xA9, 0x99, 0x70,
0x49, 0x92, 0xFC, 0xFA, 0x70, 0x24, 0x6F, 0x7B, 0x1D, 0x93, 0x8D, 0x7D, 0xB4, 0xAE, 0x2A, 0x7D, 0x53,
0x5C, 0x68, 0xEA, 0xFA, 0x94, 0x58, 0x54, 0x28, 0xCF, 0x23, 0xFB, 0x70, 0x80, 0x7F, 0xF0, 0x4F, 0x2A,
0x0B, 0x94, 0xD7, 0x3E, 0x7F, 0x78, 0x45, 0xFC, 0xE3, 0xA9, 0x3E, 0x1E, 0x23, 0xA3, 0x7E, 0x06, 0x00,
0x1D, 0x66, 0x50, 0x9D, 0xD1, 0x1F, 0x65, 0x7E, 0x76, 0x8F, 0x47, 0x73, 0xF0, 0xAA, 0x3A, 0xC5, 0xB8,
0xB0, 0x65, 0xDD, 0x34, 0x48, 0x80, 0x30, 0x46, 0xE0, 0x0A, 0xDD, 0x1B, 0xC6, 0xD6, 0x88, 0xFB, 0x76,
0x0A, 0xA5, 0xE9, 0xB5, 0xC8, 0xBC, 0x0B, 0x82, 0x1C, 0x33, 0xA3, 0x4D, 0xD3, 0xCE, 0x2F, 0x2A, 0x8E,
0xFA, 0xAA, 0xB2, 0x5D, 0x57, 0x89, 0x03, 0x56, 0x5F, 0xF2, 0x05, 0xF7, 0x24, 0xE6, 0xB6, 0x13, 0x84,
0xBC, 0x5D, 0xA5, 0x8F, 0x0D, 0xAC, 0xC1, 0xA7, 0xDB, 0x2A, 0xDF, 0xB9, 0xDA, 0x91, 0xFB, 0xF1, 0xD7,
0x83, 0x36, 0xCC, 0x3D, 0xBE, 0x14, 0xEF, 0x51, 0x57, 0xE1, 0xBF, 0x6A, 0x3F, 0x5F, 0xEA, 0xA8, 0x08,
0xB6, 0x83, 0x84, 0xA2, 0x8B, 0x2F, 0x13, 0x2B, 0x59, 0x9D, 0x86, 0x29, 0x22, 0x53, 0x17, 0xEE, 0x15,
0x84, 0x3B, 0x1E, 0x2D, 0x10, 0xF0, 0x8B, 0xC3, 0xAD, 0x4B, 0x45, 0x50, 0x06, 0x12, 0xAA, 0x94, 0x60,
0x07, 0x09, 0x6B, 0x2A, 0xDA, 0xBF, 0x86, 0x90, 0x9A, 0xFB, 0xAF, 0xEC, 0xBE, 0x05, 0x4A, 0x1E, 0xFC,
0x6E, 0xFE, 0x81, 0xC0, 0x1B, 0xB2, 0x39, 0x2F, 0x5C, 0x05, 0x40, 0xAB, 0x0B, 0x4E, 0xE3, 0x69, 0x15,
0x9A, 0x3F, 0x70, 0x94, 0x10, 0xE7, 0x91, 0xEF, 0x1E, 0x69, 0xE3, 0x6D, 0xF6, 0x43, 0xE5, 0xEB, 0xE4,
0x1E, 0xFE, 0xAC, 0xAF, 0x64, 0xCD, 0xF0, 0x59, 0x07, 0x0B, 0x5A, 0xC8, 0xED, 0x19, 0x84, 0xD0, 0x4D,
0xF8, 0xCC, 0x85, 0x54, 0x75, 0xFC, 0xE8, 0x6E, 0x3F, 0x5E, 0xF8, 0xB6, 0x39, 0xDC, 0x7F, 0x24, 0x7D,
0x7E, 0x83, 0x3F, 0xF4, 0xB9, 0x8A, 0xE8, 0xC8, 0xDC, 0x7A, 0xFB, 0x2E, 0x63, 0xB2, 0x5C, 0x11, 0xF6,
0x8B, 0xFB, 0x83, 0x20, 0xBA, 0x00, 0x9A, 0x04, 0xFB, 0xD5, 0xDD, 0x51, 0x8D, 0x90, 0x59, 0x0C, 0xF9,
0xA5, 0xEF, 0x24, 0x98, 0x09, 0x26, 0x97, 0x32, 0x97, 0xBC, 0xAA, 0x3D, 0x82, 0x9E, 0xB0, 0x2D, 0xA8,
0x23, 0x76, 0xCE, 0x5C, 0x59, 0x2A, 0x9F, 0x12, 0xAA, 0x60, 0x8D, 0x8F, 0x1F, 0xD1, 0xE1, 0x67, 0xC8,
0x2A, 0x19, 0x67, 0x66, 0x5B, 0x81, 0xAE, 0x98, 0xD3, 0x3A, 0x17, 0xF5, 0xAA, 0xD6, 0x43, 0xF7, 0xD9,
0x6A, 0xA4, 0x71, 0x08, 0xFC, 0xFE, 0x5F, 0x01, 0xA4, 0x26, 0x06, 0x94, 0x95, 0xBB, 0xBE, 0x0A, 0xCF,
0x2D, 0x88, 0x1F, 0x7E, 0xBF, 0x21, 0x12, 0x51, 0x1E, 0xBC, 0xB9, 0xA3, 0x56, 0x20, 0x9C, 0x60, 0x82,
0x57, 0x41, 0x82, 0xCC, 0x91, 0xC8, 0xFF, 0xEF, 0xCD, 0xCF, 0x61, 0x17, 0xE5, 0xE9, 0x55, 0xA0, 0xFD,
0xD4, 0x12, 0x1B, 0x5C, 0xCA, 0x75, 0x73, 0x19, 0x87, 0xD6, 0xD6, 0x08, 0x29, 0xEE, 0xA9, 0x96, 0xFE,
0x7F, 0x6A, 0xBA, 0x68, 0xE9, 0x88, 0x3F, 0xD7, 0x6B, 0xE1, 0x9C, 0x26, 0x45, 0x39, 0x28, 0x5B, 0xC1,
0xED, 0x40, 0xF3, 0x1C, 0x1E, 0x05, 0xC3, 0x69, 0x29, 0x7A, 0xF1, 0x48, 0xDA, 0xB3, 0xB3, 0xF0, 0x86,
0xC9, 0xCE, 0xDD, 0x29, 0xDA, 0x53, 0xF5, 0x47, 0x1A, 0x11, 0x5E, 0x07, 0x5A, 0x94, 0x7C, 0x72, 0x21,
0x71, 0x63, 0xAE, 0xB3, 0xEC, 0x17, 0xA8, 0xC4, 0xDB, 0x13, 0x61, 0x58, 0xA4, 0x6C, 0x63, 0x0A, 0xA6,
0xD5, 0xC5, 0xFF, 0x0E, 0xC3, 0x3B, 0xCB, 0xA2, 0x56, 0x04, 0x86, 0x32, 0x71, 0xBF, 0xD9, 0xE5, 0xED,
0x01, 0x52, 0xC8, 0xD3, 0x2D, 0x08, 0xF9, 0x6B, 0xF0, 0x53, 0x71, 0x23, 0x07, 0xA7, 0xDD, 0xA1, 0xA1,
0x39, 0xA8, 0x27, 0x7C, 0xAD, 0xCE, 0xBA, 0x46, 0xDE, 0xEF, 0x5C, 0x8C, 0x98, 0xBE, 0xDA, 0xAE, 0x63,
0xF6, 0xDF, 0x4C, 0x7F, 0x29, 0x83, 0x65, 0x87, 0x09, 0x2B, 0xFB, 0x10, 0xC1, 0xDB, 0xFF, 0x08, 0x2A,

```
0x9D, 0x87, 0x29, 0x86,   0x34, 0x0E, 0xA3, 0x43, 0x29, 0x46, 0x33, 0xF0, 0x6C, 0x53,   0x20, 0x89, 0x36,
0x49, 0x7E, 0x5B, 0x11, 0x80, 0xA6, 0x48,   0x80, 0xB9, 0xB9, 0x32, 0xB3, 0xC8, 0x16, 0xD2, 0x05, 0x47,
0x53, 0xB5, 0x96, 0x15, 0x82, 0x16, 0x3B, 0x25, 0x47, 0x53,   0x3E, 0x95, 0xEA, 0xAD, 0x9D, 0x91, 0x94,
0xF9, 0xD4, 0x8B,   0x53, 0x66, 0xAE, 0x8C, 0x0E, 0x1F, 0xB0, 0xCF, 0xA4, 0x3E,   0x91, 0x9A, 0xE3, 0xDE,
0xB5, 0xB0, 0xDA, 0xF2, 0x9D, 0x0C,   0x79, 0xB6, 0x7B, 0x70, 0x0A, 0x50, 0x4F, 0x7A, 0x58, 0x33,   0x89,
0x74, 0x9D, 0xA7, 0xAD, 0x71, 0x9C, 0xD0, 0x8F, 0xCA,   0xC9, 0x51, 0xC7, 0x81, 0x0C, 0xC6, 0x6A, 0x8A,
0x52, 0xB6,   0x0C, 0xD9, 0x86, 0x92, 0x33, 0xDC, 0x9E, 0xA9, 0xE7, 0xF2,   0xED, 0xA5, 0x4A, 0x80, 0x5F,
0x00, 0xB7, 0xDB, 0x75, 0xA9,   0x81, 0x12, 0x56, 0xC7, 0xE8, 0x72, 0xCB, 0xC9, 0x62, 0x38,   0x03, 0x76,
0xB2, 0x57, 0xCD, 0x1A, 0xF7, 0xFF, 0x3C, 0x1D,   0x5F, 0xB4, 0x4C, 0x90, 0x3E, 0x8D, 0x10, 0x7E, 0x33,
0xFD,   0x59, 0xD9, 0xAD, 0xF5, 0x33, 0x58, 0x41, 0xFF, 0xDA, 0x8E,   0x06, 0x37, 0x52, 0x9E, 0x68, 0xFC,
0xCC, 0x59, 0xAA, 0x27,   0x11, 0x34, 0x02, 0x63, 0x00, 0x03, 0x06, 0x60, 0x90, 0xDE,   0x07, 0xD9, 0x15,
0x8A, 0x71, 0x03, 0x6C, 0x6F, 0x4A, 0x56,   0x8F, 0x08, 0x7F, 0x63, 0xE0, 0xA9, 0x23, 0x5B, 0x27, 0xE8,
0xD7, 0xC0, 0x8E, 0xD6, 0xA0, 0x6F, 0xB5, 0x1D, 0x96, 0x39,   0x21, 0x76, 0x3C, 0x74, 0xDC, 0xA2, 0xC9,
0x3A, 0xCC, 0x1B,   0x67, 0x06, 0x1C, 0xF6, 0x48, 0xF4, 0x57, 0x31, 0x48, 0xF2,   0x07, 0xD7, 0xCF, 0xF7,
0x63, 0x50, 0xC0, 0x03, 0x15, 0x2E,   0xA0, 0x26, 0x48, 0xA6, 0x2F, 0x3F, 0xD2, 0x96, 0x0A, 0xEE,   0x52,
0x1F, 0xBF, 0x1A, 0x0F, 0xB8, 0xAF, 0x32, 0xBC, 0x78,   0x46, 0x43, 0x36, 0x28, 0x30, 0x4E, 0x7B, 0x57,
0x0D, 0x58,   0xB5, 0xB6, 0x2E, 0x3D, 0x9B, 0x32, 0xCA, 0x1C, 0x69, 0x74,   0x42, 0x13, 0xD3, 0x4F, 0x64,
0x2E, 0x1B, 0x65, 0xBB, 0x0A,   0x1F, 0xAA, 0xD9, 0x5E, 0xBD, 0x2F, 0xA0, 0xD3, 0xA8, 0xEF,   0x1B, 0xAC,
0xF8, 0x42, 0x96, 0x6F, 0xC3, 0x44, 0x6E, 0x2F,   0x97, 0x36, 0x9A, 0x18, 0x1E, 0x0D, 0xB9, 0xA0, 0x29,
0x5D,   0xCB, 0xD4, 0xE2, 0xBE, 0x55, 0x59, 0xA6, 0x26, 0x9E, 0x57,   0xAA, 0x62, 0xEB, 0xC0, 0x6D, 0x76,
0x45, 0x80, 0xC4, 0xDF,   0x91, 0x39, 0x32, 0xE9, 0xC3, 0xFD, 0x94, 0x49, 0xCE, 0x8C,   0x98, 0xDE, 0x4A,
0x6D, 0x6E, 0x60, 0xE7, 0x8D, 0x89, 0x95,   0x26, 0x8B, 0x79, 0x53, 0xBC, 0xB2, 0xFB, 0xC6, 0x9C, 0x9B,
0x99, 0xE2, 0x99, 0xE2, 0xAE, 0xB8, 0x94, 0x62, 0x9B, 0x3F,   0x41, 0x5F, 0xC9, 0x4E, 0x64, 0xDC, 0x93,
0xFA, 0xB8, 0x0B,   0x9E, 0x6E, 0x2F, 0x92, 0xD9, 0xDB, 0xCF, 0xFA, 0x85, 0x9F,   0x0E, 0xB0, 0x54, 0x72,
0x4A, 0x3D, 0xFA, 0x48, 0x10, 0xFE,   0x14, 0x4D, 0x6F, 0xA2, 0x65, 0x80, 0xF1, 0x86, 0xE3, 0x37,   0x28,
0x6B, 0x7D, 0x7F, 0xF0, 0x62, 0xCF, 0x8E, 0x66, 0x3E,   0xE3, 0x65, 0xDD, 0x26, 0xDE, 0xA4, 0x0D, 0x6D,
0x26, 0x1C,   0x5D, 0x69, 0x70, 0xBE, 0x99, 0xE2, 0xD1, 0xDB, 0xDE, 0xC2,   0x90, 0xF5, 0xB1, 0x69, 0x2E,
0x75, 0x3C, 0xB1, 0xA5, 0x93,   0xF8, 0x01, 0x40, 0xE7, 0x39, 0x42, 0x0C, 0x39, 0xE0, 0xED,   0x97, 0xC3,
0xBA, 0x89, 0x77, 0xC3, 0xB6, 0x5E, 0xA8, 0x40,   0xF6, 0x8F, 0x32, 0xB3, 0x23, 0x9E, 0x92, 0xDB, 0x10,
0xB2,   0xD0, 0xFD, 0xB4, 0x32, 0x2E, 0xB3, 0xC6, 0x24, 0x6F, 0xCE,   0x01, 0xCE, 0x27, 0xD8, 0x5C, 0x7D,
0xA5, 0x1F, 0xCC, 0x48,   0x53, 0x07, 0x8F, 0x8B, 0x53, 0xAD, 0x94, 0xBA, 0xE7, 0x62,   0xEB, 0x53, 0xEA,
0xEC, 0xA0, 0x05, 0x94, 0x0C, 0xD4, 0x72,   0x6D, 0x24, 0x50, 0xC1, 0x85, 0xA3, 0xBB, 0x51, 0x52, 0x13,
0xCF, 0xF3, 0x39, 0x3F, 0x5B, 0x5A, 0x6D, 0xBD, 0xB6, 0x9B,   0xAE, 0x4C, 0x60, 0x1A, 0x9C, 0x48, 0x40,
0x6E, 0x0A, 0xC5,   0x96, 0x25, 0xCE, 0x0A, 0x26, 0x9A, 0x0E, 0x47, 0xAD, 0xC8,   0x43, 0x0C, 0xD7, 0xF8,
0xB7, 0x5B, 0xAA, 0x3B, 0x16, 0xBF,   0x8A, 0xFF, 0x7B, 0x0F, 0xF3, 0x5F, 0x0B, 0x4D, 0x62, 0xE1,   0x3C,
0x5E, 0xE0, 0x70, 0xB6, 0x31, 0xF9, 0xBF, 0xC3, 0x77,   0xDE, 0xB6, 0x17, 0xF6, 0x0E, 0x53, 0x32, 0x3E,
0x3F, 0x93,   0x73, 0xE7, 0x72, 0xCE, 0x8D, 0xC3, 0xFE, 0x89, 0xEF, 0xD7,   0xCA, 0xEA, 0x85, 0xB2, 0xF0,
0xF2, 0xB8, 0x7B, 0x46, 0xB7,   0x71, 0x98, 0x79, 0x8B, 0xAC, 0x0B, 0xDA, 0x4C, 0x86, 0x7B,   0x42, 0x53,
0x69, 0x05, 0x6B, 0xDA, 0x34, 0x4B, 0xB3, 0xB2,   0x49, 0x2D, 0x9D, 0xAB, 0xB9, 0xC8, 0x2B, 0x3F, 0xB3,
0x9D,   0x66, 0x71, 0xD0, 0x9F, 0xFC, 0x4E, 0xF0, 0xCE, 0x4F, 0xAC,   0x4E, 0x08, 0x2A, 0x23, 0xDE, 0xA2,
0x1F, 0x2F, 0x21, 0xCE,   0x73, 0x42, 0xB6, 0xF2, 0xEF, 0x4B, 0x6E, 0x56, 0xF6, 0x35,   0xAD, 0x2D, 0x61,
0x7D, 0x44, 0xCB, 0x61, 0x08, 0xAF, 0xD3,   0x94, 0x12, 0x7B, 0x61, 0x48, 0xDE, 0x2E, 0xB8, 0x98, 0xF8,
0xC3, 0x66, 0xBC, 0x27, 0x31, 0x77, 0x33, 0x9F, 0xB7, 0x68,   0x39, 0xB8, 0x7C, 0x16, 0x76, 0x68, 0xA9,
0x58, 0x08, 0xE6,   0x07, 0xFB, 0xBD, 0xEF, 0x27, 0xCD, 0x47, 0x71, 0xCD, 0xCB,   0x81, 0x48, 0x4B, 0xCC,
0xA5, 0x85, 0xD2, 0xDA, 0xC9, 0x8C,   0x5C, 0x68, 0xC3, 0xA6, 0x83, 0x98, 0x6B, 0xEE, 0x51, 0x8A,   0x65,
0x32, 0x94, 0x27, 0x11, 0x2C, 0x6D, 0xA3, 0x6A, 0xD3,   0xF6, 0xD5, 0xBB, 0x27, 0xBA, 0x54, 0x1C, 0x92,
0xF7, 0xBF,   0x17, 0x7F, 0x7A, 0xA7, 0x01, 0x8B, 0x84, 0x56, 0x46, 0x13,   0xCF, 0x18, 0xD1, 0x60, 0xC8,
0x08, 0xE0, 0x3C, 0x63, 0x2F,   0x4F, 0xFA, 0xE8, 0x5C, 0x3B, 0xAD, 0x5C, 0x45, 0x62, 0x3A,   0xD1, 0xBE,
0x75, 0x3D, 0x79, 0x26, 0xF0, 0xA2, 0x82, 0x23,   0xB8, 0x8C, 0xFE, 0xC7, 0x2A, 0x38, 0x03, 0xC1, 0x6D,
0x87,   0xFD, 0xBA, 0x28, 0x55, 0x22, 0xE7, 0x4F, 0xB4, 0x33, 0xB7,   0x7D, 0x88, 0xAE, 0x79, 0x4F, 0x87,
0x0F, 0xE3, 0x26, 0xD2,   0xE7, 0x4E, 0xC8, 0x69, 0xAB, 0x8A, 0x15, 0x19, 0x95, 0xC3,   0x0D, 0x57, 0xD3,
0x5B, 0x67, 0x24, 0x10, 0x31, 0x35, 0x23,   0xA5, 0xDF, 0x0B, 0xC7, 0xD3, 0x20, 0x11, 0x8B, 0xB3, 0x09,
0xD3, 0x3C, 0x6B, 0x25, 0x80, 0xAE, 0xCD, 0x50, 0x32, 0x19,   0xC0, 0x09, 0xA8, 0x52, 0x93, 0x0A, 0x78,
0x8F, 0x01, 0x0A,   0xD2, 0x24, 0x96, 0x52, 0x06, 0x2A, 0xBD, 0xD5, 0x71, 0x42,   0x5D, 0xB5, 0x23, 0x22,
0xBA, 0xA5, 0x17, 0xAB, 0xA0, 0xE3,   0x2B, 0xB5, 0x34, 0xCC, 0x83, 0x98, 0xAC, 0x23, 0x92, 0xE7,   0x7F,
0x3B, 0x6B, 0x8A, 0x29, 0x8F, 0x44, 0x6D, 0x07, 0x67,   0xA7, 0xAA, 0x1B, 0x37, 0xE1, 0x2B, 0xE5, 0x39,
0x7E, 0x42,   0xEB, 0xFA, 0x2C, 0x09, 0x1D, 0x77, 0x95, 0xAB, 0x3A, 0x41,   0x4B, 0xD2, 0x73, 0xAF, 0xE2,
0xC8, 0xA3, 0xEA, 0xFE, 0xAE,   0x69, 0x75, 0x3F, 0x54, 0x93, 0x40, 0x13, 0x7A, 0xC8, 0xEA,   0x3B, 0x85,
0xD1, 0x82, 0xDD, 0x6B, 0x93, 0xAD, 0xB1, 0x9A,   0xD5, 0x33, 0x8B, 0xD9, 0x3F, 0x40, 0x8E, 0x4E, 0xCA,
0xF1,   0x74, 0x58, 0xFC, 0xD7, 0xA7, 0xC7, 0xBF, 0x6D, 0x61, 0x13,   0x9F, 0x64, 0x91, 0x1E, 0xC4, 0x00,
```

```
0x36, 0x3B, 0xB5, 0x66,  0xCF, 0xD6, 0xD0, 0x85, 0x1B, 0xDB, 0xB7, 0x94, 0x8F, 0xAF,  0x08, 0x2D, 0x28,
0xBD, 0xF9, 0x8C, 0x60, 0x6C, 0xC9, 0x93,  0x10, 0x0F, 0x0E, 0x73, 0x99, 0xFD, 0xDA, 0x7E, 0xE0, 0xA1,
0xC4, 0xE8, 0xE6, 0x19, 0x65, 0x80, 0x98, 0xB3, 0xA7, 0xC1,  0x8E, 0x2C, 0xA4, 0x2B, 0xC5, 0xAB, 0x6E,
0xAD, 0x3B, 0xF5,  0xA6, 0xC1, 0x6D, 0x1D, 0x1A, 0xBD, 0x3E, 0xB8, 0xE5, 0xAA,  0x9A, 0x7D, 0xD4, 0x56,
0x0E, 0x12, 0x33, 0x8E, 0xBF, 0x18,  0x5B, 0x4B, 0x17, 0x66, 0x76, 0x3E, 0x01, 0xC7, 0x73, 0x07,  0xF8,
0x40, 0xD6, 0x93, 0x97, 0xB5, 0x31, 0x25, 0xD1, 0xAA,  0x00, 0xF9, 0x3C, 0x42, 0x93, 0x77, 0x54, 0x11,
0x54, 0x71,  0x2E, 0x09, 0x77, 0xE1, 0x10, 0x58, 0x53, 0xCF, 0xB3, 0xD2,  0xB2, 0x72, 0x60, 0x89, 0x18,
0xAD, 0xFC, 0x09, 0xF5, 0xBC,  0x68, 0x01, 0xC2, 0xF9, 0x35, 0xE3, 0x7E, 0xB7, 0x5C, 0xE5,  0x3B, 0x9D,
0x01, 0x8C, 0xD5, 0x6B, 0x91, 0xEA, 0x9F, 0x51,  0x29, 0xD6, 0xCD, 0x2E, 0x67, 0xE8, 0x19, 0x49, 0x27,
0xEE,  0x12, 0xFC, 0x2F, 0x46, 0x0E, 0xF9, 0xCA, 0x35, 0x54, 0x67,  0x08, 0xB6, 0xED, 0x06, 0x25, 0xFF,
0x28, 0x7E, 0xCA, 0x4D,  0xBD, 0x8C, 0x76, 0x7D, 0x23, 0x8D, 0xF4, 0xAF, 0x77, 0x6C,  0x46, 0x21, 0x64,
0xF2, 0x5F, 0x7A, 0x51, 0xA5, 0xCD, 0x87,  0xA8, 0xF4, 0x63, 0x81, 0x17, 0xDB, 0x21, 0x34, 0x8E, 0x3D,
0xB1, 0xDB, 0x96, 0x25, 0xFF, 0xCE, 0xAE, 0x7D, 0xB5, 0xB8,  0x01, 0x90, 0xF4, 0x07, 0xCB, 0xFA, 0x50,
0xDB, 0xA8, 0xE3,  0xC9, 0x3F, 0xB4, 0x98, 0x53, 0xFE, 0x43, 0x8F, 0x2C, 0x9D,  0xB9, 0xF3, 0x92, 0x5D,
0x86, 0x3F, 0x8B, 0x82, 0xD0, 0x97,  0x32, 0xBF, 0x23, 0x86, 0xEC, 0x3C, 0xF3, 0x56, 0x29, 0xD5,  0x5C,
0xEB, 0x50, 0x39, 0xB8, 0x88, 0x97, 0x70, 0xE3, 0xE0,  0xDA, 0x3E, 0x61, 0x03, 0x1F, 0xC4, 0x26, 0x07,
0x6F, 0x00,  0x18, 0x89, 0x29, 0x0F, 0xF4, 0x08, 0xFD, 0x84, 0xBA, 0x52,  0xF6, 0xAB, 0x4A, 0xDF, 0x50,
0x6D, 0xB0, 0x5E, 0x5C, 0x6F,  0xD8, 0xB6, 0x0A, 0x9A, 0x42, 0x25, 0x75, 0xB2, 0x5E, 0x7C,  0x6A, 0x21,
0xD4, 0x63, 0xF0, 0xC6, 0xA1, 0x02, 0xEC, 0x28,  0x1E, 0xCC, 0x73, 0x71, 0x75, 0xD5, 0x0F, 0x4F, 0xE1,
0xE4,  0x11, 0x24, 0x6B, 0x79, 0x7D, 0x12, 0xC7, 0xB3, 0xED, 0xED,  0x93, 0x98, 0x63, 0xFF, 0x34, 0x6E,
0xFC, 0x36, 0x43, 0x83,  0x62, 0x9A, 0x64, 0x0A, 0xF3, 0x94, 0xE1, 0xC5, 0x00, 0xCA,  0x01, 0x4B, 0xCE,
0x3F, 0x48, 0xB7, 0x57, 0x69, 0x87, 0x9A,  0x82, 0xC8, 0xC4, 0xA8, 0xAD, 0x2E, 0x68, 0xBF, 0x1E, 0x85,
0xB1, 0x83, 0x4F, 0x1D, 0x39, 0x8A, 0x36, 0x04, 0xDD, 0xDB,  0x06, 0x2F, 0xFA, 0xF6, 0xF7, 0xEC, 0x7C,
0x16, 0x22, 0x17,  0x7B, 0x12, 0x28, 0xAA, 0xD8, 0x78, 0xE2, 0xF3, 0x23, 0x83,  0x1B, 0x6C, 0xCC, 0xD6,
0x3D, 0xA0, 0x99, 0x22, 0x3A, 0x85,  0xA8, 0x84, 0xD1, 0xBA, 0x26, 0x1D, 0x70, 0x01, 0x34, 0x94,  0x3D,
0x1F, 0x0C, 0xC5, 0x12, 0xD8, 0xCC, 0x55, 0x74, 0xBE,  0xB3, 0xC3, 0x4B, 0xE5, 0x45, 0x3A, 0x46, 0x17,
0x2E, 0x5F,  0x43, 0xE5, 0x0F, 0x29, 0xA6, 0x39, 0x04, 0x5E, 0xEA, 0x07,  0x1F, 0x10, 0xBB, 0x77, 0xB1,
0xD2, 0xB7, 0xBF, 0xDA, 0x30,  0x3B, 0x7C, 0x14, 0x9E, 0x22, 0xA6, 0x29, 0xBD, 0xF2, 0xB4,  0xBF, 0xCC,
0x13, 0x79, 0xB2, 0xE7, 0xA0, 0x3C, 0x81, 0x33,  0xE1, 0xB8, 0x40, 0x95, 0x5B, 0xCD, 0x6E, 0x1E, 0xDB,
0x7E,  0x52, 0x77, 0xD1, 0xBC, 0x80, 0x31, 0x40, 0x86, 0x7A, 0xD7,  0xB6, 0x5B, 0x87, 0xE6, 0xE3, 0xC5,
0xBD, 0x30, 0x6B, 0x2E,  0xFA, 0x19, 0x7D, 0x41, 0xF1, 0x73, 0x90, 0xE6, 0x53, 0x58,  0x1A, 0x88, 0x48,
0x9A, 0x83, 0x83, 0x81, 0x25, 0xEE, 0xDC,  0xDD, 0x11, 0xCD, 0x22, 0x66, 0x41, 0x84, 0x27, 0x65, 0xC6,
0x75, 0x8F, 0x78, 0x98, 0x36, 0x31, 0x30, 0x1B, 0xB4, 0xBD,  0x4B, 0xC1, 0x23, 0x73, 0x93, 0x00, 0x91,
0x8A, 0xD1, 0x39,  0x98, 0x27, 0x77, 0xC0, 0xFA, 0x21, 0x15, 0x17, 0xB3, 0xD6,  0x89, 0xDB, 0x7C, 0xE2,
0xEA, 0x7A, 0x2B, 0xAE, 0xA4, 0x1D,  0x24, 0x17, 0xD3, 0xD5, 0x4F, 0xEC, 0x3C, 0x9B, 0x06, 0xA1,  0xFD,
0xD6, 0xCD, 0xAD, 0x37, 0x95, 0xFA, 0x23, 0x77, 0x54,  0x64, 0x7C, 0x2F, 0x95, 0x02, 0x26, 0x6A, 0x4A,
0xAA, 0xFC,  0xE4, 0xF9, 0x49, 0xCA, 0x27, 0xFD, 0xFF, 0x10, 0xE2, 0xE1,  0xB4, 0xD9, 0x50, 0xC2, 0xC4,
0x89, 0xD6, 0x5C, 0x44, 0x68,  0xEA, 0xD3, 0xBB, 0x4A, 0xD6, 0x33, 0x3E, 0x42, 0xB3, 0x23,  0x69, 0x05,
0x2A, 0x9B, 0x1D, 0xDC, 0x81, 0x1C, 0xA9, 0x8A,  0x47, 0x2F, 0x84, 0x3D, 0x4E, 0x84, 0x72, 0x50, 0xAF,
0x23,  0xF3, 0x63, 0xCE, 0x26, 0xB3, 0xD6, 0xFF, 0xB7, 0x9D, 0x16,  0x8D, 0x5C, 0x6D, 0xF7, 0x5C, 0x6E,
0x7B, 0x1D, 0x8E, 0x26,  0xC0, 0xFE, 0x8C, 0x2D, 0x8E, 0x5F, 0xC5, 0xA0, 0x90, 0xCE,  0xF5, 0xA4, 0x08,
0x06, 0x0A, 0x9F, 0x34, 0xAC, 0xDA, 0xA0,  0xC7, 0x71, 0x2E, 0x12, 0x98, 0x00, 0x5C, 0x40, 0xDD, 0x1A,
0xE2, 0xC2, 0x59, 0x56, 0xF3, 0x5E, 0xE8, 0x64, 0x6F, 0x0D,  0xA2, 0xD5, 0x21, 0x50, 0x9C, 0x8B, 0x54,
0x88, 0x01, 0xA6,  0xA0, 0x58, 0x55, 0xF9, 0x57, 0xD2, 0x63, 0x13, 0x43, 0x97,  0xC3, 0x8A, 0xC1, 0xC8,
0xA2, 0xCC, 0xCE, 0xC7, 0x8E, 0xBF,  0x1F, 0x58, 0x8F, 0x2A, 0x19, 0xBC, 0x6E, 0x07, 0x91, 0x50,  0x23,
0xEB, 0x25, 0xBC, 0x90, 0xCB, 0x88, 0x8A, 0xA2, 0x06,  0x8C, 0xC6, 0x30, 0xC7, 0xCC, 0x04, 0x93, 0xF6,
0xB4, 0x74,  0x52, 0x76, 0x86, 0x79, 0xC5, 0x60, 0x98, 0xDD, 0x29, 0x46,  0x4A, 0x4B, 0x10, 0x1D, 0x35,
0x81, 0xE7, 0x59, 0xE8, 0xA1,  0x90, 0xDD, 0x75, 0x5C, 0x36, 0xB1, 0x51, 0x22, 0xE2, 0xF7,  0xF8, 0xE8,
0xDB, 0xD9, 0x4A, 0xAD, 0x08, 0xD9, 0x35, 0xF8,  0x00, 0xC4, 0x34, 0x39, 0x03, 0xC8, 0x37, 0xC5, 0x60,
0x3D,  0x25, 0x7E, 0x07, 0xBE, 0x25, 0x27, 0xB7, 0x86, 0x3A, 0x3A,  0x8C, 0xB2, 0xC1, 0xD4, 0x4E, 0xA9,
0x68, 0x15, 0x55, 0xB8,  0xBD, 0xBA, 0xFF, 0x0F, 0xD3, 0x63, 0x63, 0x9E, 0xED, 0x1E,  0x48, 0xAB, 0x18,
0xEA, 0x7D, 0xAD, 0x38, 0xD2, 0xE9, 0x77,  0x1B, 0x4B, 0xDD, 0xD9, 0x78, 0x3C, 0x27, 0x47, 0xFD, 0x02,
0xAD, 0xFE, 0x38, 0x45, 0xB5, 0xA0, 0xCC, 0x2A, 0xBD, 0xAD,  0x5B, 0x53, 0xFD, 0xA5, 0x50, 0x42, 0x5D,
0x60, 0xE9, 0x51,  0x2C, 0x4A, 0x8D, 0x58, 0xD2, 0x2B, 0x41, 0x95, 0x69, 0x3C,  0xDD, 0xD1, 0xAA, 0x9F,
0xBA, 0x41, 0x72, 0x40, 0x27, 0xC1,  0x7E, 0x38, 0xEE, 0x51, 0xC2, 0x06, 0x61, 0x14, 0x3A, 0xC4,  0xBC,
0x4C, 0x58, 0x23, 0x42, 0xC0, 0x6E, 0x70, 0x2E, 0x36,  0x2F, 0x8E, 0xD5, 0x3E, 0x9B, 0x57, 0x7F, 0x7D,
0xD7, 0x1D,  0x6D, 0x56, 0x1B, 0x52, 0xE2, 0x5B, 0xF8, 0x99, 0xCE, 0xAE,  0xD8, 0x51, 0xFB, 0xA0, 0xB8,
0xA1, 0xE7, 0x03, 0x45, 0x0F,  0xCF, 0xEA, 0xE8, 0x8A, 0x15, 0xAC, 0x59, 0xC3, 0x91, 0x49,  0x7C, 0x83,
0xB0, 0x13, 0x43, 0x51, 0x49, 0x2C, 0xE4, 0x33,  0x0B, 0x84, 0xE8, 0x5B, 0x9E, 0x82, 0x95, 0x49, 0x1B,
```

```
0x76,   0x0B, 0x87, 0x56, 0x36, 0xBB, 0x2E, 0xDC, 0xE0, 0x13, 0xF1,   0xE1, 0x91, 0x11, 0x40, 0x46, 0xA3,
0x8E, 0x6B, 0x0B, 0xC2,   0x19, 0xE0, 0x2B, 0x32, 0x7C, 0x81, 0x22, 0x12, 0xE9, 0xE0,   0x58, 0x05, 0x08,
0x56, 0x46, 0x83, 0xD8, 0xB9, 0x9B, 0x3F,   0xBE, 0xC6, 0x3B, 0x43, 0x6F, 0x57, 0x17, 0x8E, 0xDE, 0x21,
0x25, 0x9E, 0x2C, 0xD3, 0x10, 0xC2, 0x9B, 0x47, 0xAF, 0xB4,   0xD3, 0xDD, 0x05, 0xD8, 0x0C, 0xF2, 0x69,
0x9A, 0x33, 0xB1,   0xFD, 0x1E, 0xEB, 0x3F, 0x4C, 0x5B, 0xCD, 0x22, 0x38, 0xB5,   0x80, 0xC0, 0x88, 0xDD,
0x9A, 0xB5, 0xF6, 0xB5, 0x63, 0x13,   0x45, 0x70, 0xF4, 0xD8, 0x39, 0x59, 0x5E, 0xBE, 0x02, 0x0D,   0xB6,
0xC7, 0x43, 0x43, 0x4F, 0x49, 0xF1, 0xA6, 0x3C, 0xDD,   0x5F, 0xC1, 0xF9, 0x35, 0x2D, 0xA1, 0x97, 0xC7,
0x3F, 0xB6,   0xCD, 0x2F, 0x62, 0x45, 0x1F, 0xE0, 0x6D, 0x65, 0x5E, 0xFE,   0x8B, 0xF9, 0xB8, 0xE1, 0xCE,
0xF7, 0xCB, 0xDE, 0xD2, 0x55,   0x72, 0xA8, 0x26, 0xF2, 0x11, 0x2F, 0x75, 0xFA, 0x8C, 0x23,   0x60, 0xFD,
0x6F, 0x0E, 0xFD, 0xB3, 0xAD, 0x88, 0x47, 0xB7,   0x6C, 0x49, 0xE7, 0x6B, 0x76, 0x4F, 0xFB, 0xF2, 0x5B,
0x94,   0x0F, 0xB4, 0x65, 0x70, 0x84, 0x99, 0xA2, 0x0E, 0x8F, 0xBE,   0x38, 0x09, 0x01, 0x9B, 0x9D, 0x1C,
0xD7, 0xBD, 0xCB, 0x74,   0x5F, 0xFB, 0x11, 0x9B, 0xF4, 0x62, 0x8A, 0xD6, 0xBF, 0xEF,   0x94, 0x72, 0x86,
0x27, 0xCD, 0x2E, 0x36, 0x03, 0xFB, 0xDD,   0x32, 0xF4, 0x56, 0xC5, 0xD5, 0x4A, 0x68, 0x48, 0xC5, 0x28,
0x72, 0x61, 0x18, 0x10, 0xF3, 0x00, 0xA8, 0x1C, 0x45, 0xEF,   0x6D, 0x07, 0xAB, 0xDE, 0x80, 0x4A, 0xCA,
0xA3, 0xFC, 0x5A,   0x92, 0xE0, 0x78, 0x88, 0xC6, 0x4E, 0x36, 0xEE, 0x4E, 0x28,   0x05, 0xB2, 0xF7, 0xF2,
0xAC, 0xB8, 0x58, 0xF3, 0x99, 0x9D,   0x23, 0x8D, 0x41, 0x65, 0x9F, 0xEB, 0x76, 0xC0, 0x2E, 0xC6,   0x66,
0x52, 0x0E, 0x06, 0x6A, 0x38, 0x63, 0xDA, 0x2F, 0x71,   0x1B, 0xE7, 0x73, 0x96, 0x8B, 0x91, 0x33, 0x4B,
0x7C, 0x46,   0xA0, 0x9D, 0x9D, 0x3C, 0xA0, 0x20, 0x66, 0x03, 0x2B, 0x1C,   0x14, 0xED, 0x53, 0x67, 0x20,
0xF7, 0xFE, 0xB5, 0xA0, 0x3B,   0x59, 0xEE, 0x90, 0x02, 0xFB, 0x9A, 0x05, 0x47, 0xDC, 0xC6,   0x98, 0xEA,
0xCA, 0xD7, 0x09, 0x69, 0x70, 0x59, 0xB4, 0x68,   0x3C, 0xC2, 0xB6, 0x5F, 0x63, 0xEA, 0x62, 0x6F, 0x6B,
0xAC,   0x22, 0xAD, 0xB8, 0x2B, 0x36, 0x3B, 0x2B, 0xB7, 0xB8, 0x75,   0xCB, 0xCD, 0xD5, 0x3B, 0x79, 0xC7,
0x19, 0x4B, 0xF1, 0xA9,   0xB1, 0xD5, 0xC4, 0x59, 0x57, 0xAD, 0x5A, 0xA8, 0x28, 0x8E,   0xD7, 0x1E, 0x92,
0x6C, 0x01, 0x85, 0x13, 0x51, 0x62, 0x81,   0x65, 0xEA, 0x84, 0x57, 0x6F, 0x97, 0xB6, 0x0A, 0x37, 0xE0,
0x1D, 0x1E, 0x80, 0x04, 0x34, 0xC7, 0x7D, 0xBA, 0x74, 0x40,   0xD4, 0x6A, 0x72, 0xC2, 0xA1, 0x96, 0x3A,
0xF8, 0x5A, 0x9D,   0xA0, 0x50, 0xC3, 0x27, 0xF9, 0x96, 0x7F, 0x88, 0x41, 0x13,   0xE7, 0xAB, 0xAC, 0x7E,
0x77, 0xE2, 0x94, 0x67, 0x41, 0x11,   0x0D, 0xFB, 0xF2, 0x73, 0xDA, 0x18, 0x2F, 0x1C, 0xD5, 0x6B,   0xEC,
0xDE, 0x96, 0x4B, 0x83, 0x1A, 0xD6, 0xF3, 0x10, 0x9A,   0x4B, 0x8E, 0xBB, 0x2E, 0x74, 0x6D, 0x97, 0x0A,
0xCE, 0xC8,   0xC4, 0xFA, 0x4A, 0xAC, 0xB4, 0x6E, 0xDE, 0xAC, 0x58, 0xD2,   0xE1, 0x62, 0x38, 0x99, 0xAB,
0x92, 0xAE, 0xBD, 0x84, 0x52,   0x7D, 0x38, 0xFE, 0xAA, 0x6E, 0x14, 0x04, 0xA3, 0xB1, 0x72,   0xCB, 0x55,
0x97, 0x91, 0xF8, 0x31, 0x7E, 0xA9, 0x75, 0x13,   0xC0, 0xF9, 0xE2, 0x22, 0x63, 0x8F, 0xD2, 0x68, 0x3A,
0x97,   0xD7, 0x9E, 0x5B, 0xB9, 0xDE, 0xB8, 0x94, 0xA8, 0xAA, 0x34,   0x25, 0xF2, 0xC6, 0xC6, 0x81, 0xEE,
0xC8, 0x39, 0x40, 0x2B,   0x74, 0xE5, 0x52, 0x2A, 0xB9, 0x21, 0x92, 0xE8, 0x64, 0x4E,   0x24, 0x90, 0xDA,
0xD7, 0xDB, 0x67, 0x63, 0xA4, 0x8E, 0x03,   0x95, 0xD7, 0x2C, 0x87, 0x95, 0x50, 0x97, 0x8E, 0x27, 0xCC,
0x3B, 0xC7, 0x6B, 0x8E, 0x96, 0x69, 0x49, 0x07, 0x1C, 0xD1,   0x6A, 0x8E, 0x2A, 0x61, 0x26, 0xA0]
    a1 = 4284256177
    a2 = 1234567890
    a3 = 1095061718431
    v5 = 4
    flag = n2s(0x8B9551FA * a3)[::-1][:v5]
    #  print flag

    while a2:
        v3 = v5
        v5 += 1
        c = chr(((junk_data[a1 & 0xFFF] ^ a2) & 0xFF) % 128)
        #  flag += c if 0x20 <= ord(c) < 0x7f else ""
        flag += c
        #  print ord(c)
        a1 *= 77777
        a2 = a3 ^ (a2 >> 1)
        a3 >>= 1


    flag += c
    print flag
    print len(flag)
    print v5

if __name__ == "__main__":
    getInput()
```

```
getInput()
#  [a2 = 1234567890, a1 = 2136772529]
#  [a2 = 1234567890, a1 = 4284256177]
key3 = getKey3()
print "key3 -> %d" % key3
getFlag()
```

View Code

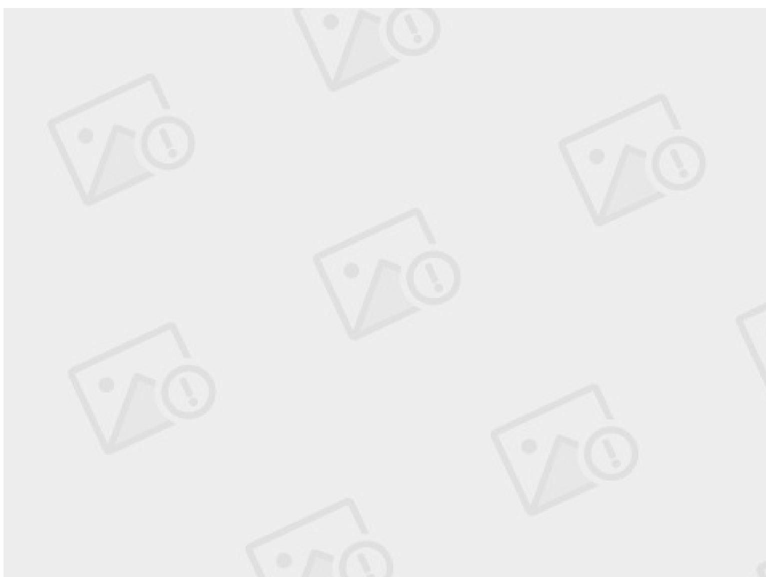
但奇怪的是用这个脚本没有跑出后几位flag,可能是python和C在边界处理上有区别

于是我换了一种方法,源程序中运算较慢是因为**calc_key3**这个函数耗费了大量的时间,把这个函数patch掉

然后在调试过程中手动给v7赋给我们计算出来的返回值,这样就可以得到flag了

0x0A mov

这个题前前后后做了好几个月，后来发现是我想麻烦了，先说一下我做这道题的历程

> IDA打开之后看到一堆mov，根据提示*MOV instruction is turing complete!*很容易google到这是movfuscator混淆，同时找到了去混淆的工具demovfuscator，安装后（不是太好装）发现去混淆的效果并不理想，生成的CG也似乎有问题
>
> 又google到类似题目的writeup，有使用qira，有使用perf，有使用pin，有直接从内存中抠出来的，因为最近在看fuzz的一些东西，所以想尝试使用pin解题目
>
> 大致了解了pin的工作原理后，安装好环境，开始找输入对输出的影响，这时偶然发现一个bug，输入n位字符时，这n位与flag的前n位相等即可，可利用这个特性爆破
>
> 
>
> 爆破脚本

```
inndy_mov [master●] cat solve.py
#!/usr/bin/env python
# -*- coding: utf-8 -*-
__Auther__ = 'M4x'

from subprocess import Popen, PIPE
from string import printable

ans = "FLAG{"
while True:
    if "}" in ans:
        print ans
        break


    for c in printable:
        f = Popen("./mov", shell = True, stdin = PIPE, stdout = PIPE)
        tmp = ans + c
        f.stdin.write(tmp + "\n")

        stdout, stderr = f.communicate()
        if "Good" in stdout:
            ans = tmp
            print ans
            break
```

View Code

实际效果也很好，不到2s就爆破出了flag

觉得使用pin的方法很巧妙，以后再补上使用pin的wp

Crypto

0x00 xor

zwhubuntu师傅告诉我这题可以用xortool解，项目地址https://github.com/hellman/xortool

主要记录一下使用方法

0x01 ffa

这个题就只想说z3大法好了。

```python
#!/usr/bin/env python3
import sympy
import json

m = sympy.randprime(2**257, 2**258)
M = sympy.randprime(2**257, 2**258)
a, b, c = [(sympy.randprime(2**256, 2**257) % m) for _ in range(3)]

x = (a + b * 3) % m
y = (b - c * 5) % m
z = (a + c * 8) % m

flag = int(open('flag', 'rb').read().strip().hex(), 16)
p = pow(flag, a, M)
q = pow(flag, b, M)

json.dump({ key: globals()[key] for key in "Mmxyzpq" }, open('crypted', 'w'))
# {"p": 240670121804208978394996710730839069728700956824706945984819015371493837551238, "q":
63385828825643452682833619835670889340533854879683013984056508942989973395315, "M":
349579051431173103963525574908108980776346966102045838681986112083541754544269, "z":
213932962252915797768584248464896200082707350140827098890648372492180142394587, "m":
282832747915637398142431587525135167098126503327259369230840635687863475396299, "x":
254732859357467931957861825273244795556693016657393159194417526480484204095858, "y":
261877836792399836452074575192123520294695871579540257591169122727176542734080}
```

View Code

程序的逻辑很简单，生成了几个大随机数**M, m, x, y, z, p, q**，通过分析代码不难得出如果得到a或者b就能得到flag，但问题就在于，怎么求出a或者b。

刚开始的想法是通过有限域的方法化简，得到a或者b，但正要动手时转念一想，题上的约束关系都很明确，可以用z3试一下，于是写了z3求解的代码：

```
def getabc():
    a = BitVec("a", 265)
    b = BitVec("b", 265)
    c = BitVec("c", 265)
    x = d["x"]
    y = d["y"]
    z = d["z"]
    m = d["m"]
    #  print x, y, z
    s = Solver()
    s.add(UGT(a, pow(2, 256, m)))
    s.add(ULT(a, pow(2, 257, m)))
    s.add(UGT(b, pow(2, 256, m)))
    s.add(ULT(b, pow(2, 257, m)))
    s.add(UGT(c, pow(2, 256, m)))
    s.add(ULT(c, pow(2, 257, m)))
    s.add(x == (a + b * 3) % m)
    s.add(y == (b - c * 5) % m)
    s.add(z == (a + c * 8) % m)

    while s.check() == sat:
        if isPrime(s.model()[a].as_long()) and isPrime(s.model()[b].as_long()) and isPrime(s.model()
[c].as_long()):
            print s.model()
        A, B, C = s.model()[a].as_long(), s.model()[b].as_long(), s.model()[c].as_long()
        s.add(Or(a != s.model()[a], b != s.model()[b], c != s.model()[c]))
    else:
        print "Finished"
        return A, B, C
```

View Code

有两点需要注意：

1. 用BitVec声明变量时，首先要估算中间变量的范围，确保运算过程中数据不会溢出(如该题中使用了较大的265位)，但为了运行速度也不能过大
2. 大整数的比较大小建议使用UGT/ULT代替>/<

本来运行的时候还替z3担心了一下会不会因为数据太大直接崩掉，但没想到用了短短的8s就跑出了结果

计算出a，b，c后，主要问题就解决了。

意识到使用flag的加密实际上是RSA时，利用a和b进行了共模攻击，很快就出flag了

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
__Auther__ = 'M4x'

from libnum import prime_test as isPrime
from libnum import n2s, xgcd, invmod
from z3 import *
```

```
d = {"p": 240670121804208978394996710730839069728700956824706945984819015371493837551238, "q":
63385828825643452682833619835670889340533854879683013984056508942989973395315, "M":
34957905143117310396352557490810898077634696610204583868198611208354175454 4269, "z":
21393296225291579776858424846489620008270735014082709889064837249218014239 4587, "m":
28283274791563739814243158752513516709812650332725936923084063568786347539 6299, "x":
25473285935746793195786182527324479555669301665739315919441752648048420409 5858, "y":
26187783679239983645207457519212352029469587157954025759116912272717654273 4080}

def getabc():
    a = BitVec("a", 265)
    b = BitVec("b", 265)
    c = BitVec("c", 265)
    x = d["x"]
    y = d["y"]
    z = d["z"]
    m = d["m"]
    #  print x, y, z
    s = Solver()
    s.add(UGT(a, pow(2, 256, m)))
    s.add(ULT(a, pow(2, 257, m)))
    s.add(UGT(b, pow(2, 256, m)))
    s.add(ULT(b, pow(2, 257, m)))
    s.add(UGT(c, pow(2, 256, m)))
    s.add(ULT(c, pow(2, 257, m)))
    s.add(x == (a + b * 3) % m)
    s.add(y == (b - c * 5) % m)
    s.add(z == (a + c * 8) % m)

    while s.check() == sat:
        if isPrime(s.model()[a].as_long()) and isPrime(s.model()[b].as_long()) and isPrime(s.model()
[c].as_long()):
            print s.model()
        A, B, C = s.model()[a].as_long(), s.model()[b].as_long(), s.model()[c].as_long()
        s.add(Or(a != s.model()[a], b != s.model()[b], c != s.model()[c]))
    else:
        print "Finished"
        return A, B, C

def getFlag((a, b, c)):
    M = d["M"]
    p = d["p"]
    q = d["q"]
    s1, s2, _ = xgcd(a, b)
    if s1 < 0:
        s1 = -s1
        p = invmod(p, M)
    elif s2 < 0:
        s2 = -s2
        q = invmod(q, M)

    flag = (pow(p, s1, M) * pow(q, s2, M)) % M
    print n2s(flag)


if __name__ == "__main__":
    #  getabc()
    getFlag(getabc())
```

View Code

整个脚本也才8s左右

> OwO多了一步求flag的过程，计算时间反而更短了，看来计算时间还是跟CPU的心情有关系

Programming

fast

ppc类型的题目，注意数据范围在int32范围内即可，同时注意如果使用pwntools写socket的话，没有必要recvuntil后再send，完全可以现将所有的输入输出都存到缓冲区中，最后一次性send，具体看脚本

```
inndy_fast [master] cat exp.py
#!/usr/bin/env python
# -*- coding: utf-8 -*-
__Auther__ = 'M4x'

from pwn import *
from numpy import int32
import time
import re
#  context.log_level = 'debug'

#  io = process("./fast")
io = remote("hackme.inndy.tw", 7707)

io.sendlineafter("the game.\n", "Yes I know")

ans = ""
res = ""
f = lambda x: int32(int(x))
for i in xrange(10000):
    n1, op, n2 = io.recvuntil("=", drop = True).strip().split(' ')
    #  print n1, op, n2
    io.recvline()

    if op == '+':
        #  print n1, op, n2
        ans = str(f(n1) + f(n2))
    if op == '-':
        ans = str(f(n1) - f(n2))
    if op == '*':
        ans = str(f(n1) * f(n2))
    if op == '/':
        ans = str(int(float(n1) / int(n2)))

    res += (ans + " ")

#  print res
io.sendline(res)
io.interactive()
io.close()
```

View Code

转载于:https://www.cnblogs.com/WangAoBo/p/hackme_inndy_writeup.html