

hswswp

Crypto

ezrsa

签到题，非常简单，只需要直接开一个模p的平方根即可，但要注意题目有问题，p和pow(flag, 2, p)位置反了，要换过来

```
1 from sympy import *
2 c=4124820799737107236308837008524397355107786950414769996181324333556950154206980
  059406402767327725312238673053581148641438494212320157665395208337575556385
3 p=1310793956350745977461620414125374748923206333620417394412326328450760432888568
  0072478669016969428366667381358004059204207134817952620014738665450753147857
4 print(sqrt_mod(c,
  p))#13040004482820526093820693618708125830699182230406913376202407698904962835203
  626640653836925
5 print(long_to_bytes(1304000448282052609382069361870812583069918223040691337620240
  7698904962835203626640653836925))
```

得到flag{9971e255f0c020e8e57fbae75f43d7fb}

hdrsa

这道题实际上是一个rsa的后门漏洞,具体的数学原理这里就不赘述,我们只需要知道当N存在因子p满足

$$4p = 427 * s^2 + 1$$

时,我们就有办法在多项式时间内对N进行分解,因此我们也就知道了题目中的列表a实际上选择了427这个数字作为D

在cryptohack上就有一道类似的题,我们直接用他提供的脚本进行对N的分解即可,当然首先得把参数换成题目的参数,具体sagemath代码如下

https://cryptohack.org/challenges/rsa_backdoor/solutions/

```
1 import sys
2 from Crypto.Util.number import *
3
4 sys.setrecursionlimit(1000000)
5 D = 427
```

6

n =

330961752887996173328854935965112588884403584531022561119743740650364958220684
640754584393850796812833007843940336784599719224428969119533284286424077547165
101460469847980799370419655082069179038497637761333327079374599506574723892143
817226751806802676013225188467403274658211563655876500997296917421904614128056
847977798146855336939306463059440416150493262973269431000762285579221126342017
624118238829230679953011897314722801993750454924627074264353692060002758521401
544361385231354313981836056855582929670811259113019012970540824951139489146393
182532414878214182086999298397377845534568556100933934481180701997394558264969
597606662342898026915506749002491326250792107348176681795942799954526068501499
100232598658650184565873243525176833451664254917655703178472944744658628534195
346977023418550761620254528178516972066618936960223660362493931786389085393392
950207048675797593816271435700130995225483316625836104802608163745376633884840
588575355936746173068655319645572100149515524131883813773486917122153248495022
372690912572541775943614626733948206252900473118240712831444072243770979419529
210034883903111038448366933374841531126421441232024514486168742686297481063089
161977054825621099768659097509939405315056325336120929492838479309609958696957
890570295444494277819063443427972643459784894450787015151715676537385237767990
406742547664321563688829289809321534752244260529319454316532580416182438749849
923354060125229328043961355894086576238519138868298499249023773237770103057707
91270972541703330906130888058398866463892828633292839968866953776989722310954
204550783825704710017434214644199415756584929214239679433211393230307782953067
246529626136446314941258877439356094775337541321331600788042698664632064112896
956898222397445497695982546922871549828242938368486774617350420790711093069910
914135319635330786253331223459637232106417577225350441291

7

e = 65537

8

c =

187275367513186345104534865239994699892170904489725413330767115192172530253625
393062151741036312498277557971553595091826062438445856091864605758318579599363
539202154625683947568962358702545878760994434813222953503460910447662183200334
960821110618746899798165363389255347363192576250804362413854445821046755759458
439443253294822553986237695607000569717855942461517564526611106601774100617668
231506539201297550376834067118784548951699927659889815770492684106287801610261
026674778509245649501695344652216367741171392139049785280654043804502329999760
613658697298671602787929199239524617160567336634126185042907593427921016129734
757065504417112269027028799047579450965076835882020261162192475637278445255805
339324893626400179818784574957669576516363342104273184813708475202313539634027
764340858242764934872804570810575764191987921655276520658100755510986290562980
055133376750812535713567917823663134974180449002466833109112866681229626239871
954125027501071383217816313440079294139254989413050731511516498127225020975071
747314764552267845933494600295296885808466296844091612401062566502515356974852
161817112538289440970059783116540091633055220150093646069438113246518726017868
258339512247175386052684861670431148484455765445960495130308147156436998327553
854387741014177421559585683382003377803158283603889312107837885491964835073892
174406797445622388505256985237867456926792546588756970045576002345376035346727
906264683596628903417932566383221754976804148878057310066885140776352202510584
461556988179369177560403923399529842871087532495739921906849249072433614545319
458973155343802539527630971239359995893495205324483418191744545506744253222956
232506980824457995662900264427265978239540089825733734306363153471606200228841
997928021468359645661221933848545854596097640552489404777927679709089475954033
350287287833943519423030861868256961619722983499902810335

9

```

10
11 def inverse(g, m):
12     u, c = g, 1
13     v, a = m, 0
14
15     while u != 0:
16         q = v.quo_rem(u)[0]
17         u, c, v, a = v - q*u, a - q*c, u, c
18
19     return a*ZZ(v).inverse_mod(n)
20
21
22 def dbl_xz(P, A, B):
23     '''
24     From http://hyperelliptic.org/EFD/g1p/auto-shortw-xz.html#doubling-dbl-2002-it-2
25     '''
26     x1, z1 = P
27
28     T1 = x1^2
29     T2 = z1^2
30     T3 = A*T2
31     T4 = T1 - T3
32     T5 = T4^2
33     T6 = B*T2
34     T7 = x1*z1
35     T8 = T6*T7
36     T9 = 8*T8
37     x3 = T5 - T9
38     T10 = T1 + T3
39     T11 = T7*T10
40     T12 = T6*T2
41     T13 = T11 + T12
42     z3 = 4*T13
43
44     return x3, z3
45
46
47 def add_xz(P1, P2, X1, A, B):
48     '''
49     From http://hyperelliptic.org/EFD/g1p/auto-shortw-xz.html#diffadd-mdadd-2002-it-3
50     '''
51     x2, z2 = P1
52     x3, z3 = P2
53
54     T1 = x2*x3
55     T2 = z2*z3
56     T3 = x2*z3
57     T4 = z2*x3
58     T5 = A*T2
59     T6 = T1 - T5

```

```

60     T7 = T6^2
61     T8 = B*T2
62     T9 = 4*T8
63     T10 = T3 + T4
64     T11 = T9*T10
65     T12 = T7 - T11
66     X5 = T12
67     T13 = T3 - T4
68     T14 = T13^2
69     Z5 = X1*T14
70
71     return x5, z5
72
73
74 def ladder(n, x0, A, B):
75     l = n.nbits()
76     x1, z1 = x0, 1
77     x2, z2 = dbl_xz((x1, z1), A, B)
78     R = [(x1, z1), (x2, z2)]
79     for i in range(2, l+1):
80         bit = (n >> (l - i)) & 1
81         R[1-bit] = add_xz(R[0], R[1], x0, A, B)
82         R[bit] = dbl_xz(R[bit], A, B)
83     return R[0]
84
85
86 def solve_challenge():
87
88     # step 1: construct class group Z/nZ[j]/(H_D(j))
89     Zn = Zmod(n)
90     Znj.<j> = Zn[]
91     H_D = j^2 + 15611455512523783919812608000*j +
155041756222618916546936832000000
92     RnD = Znj.quotient(H_D)
93     inv_den = inverse(1728 - j, H_D)
94
95     while True:
96
97         # step 2: construct elliptic curve y^2 = x^3 + A*x + B over RnD
98         r = Zn.random_element()
99         A = RnD(3*r^2*j*inv_den)
100        B = RnD(2*r^3*j*inv_den)
101
102        # step 3-4-5-6: construct P = (x0, .) a point on the curve
103        # (the other coordinate is useless thanks to Montgomery ladder
104        x0 = Zn.random_element()
105
106        # step 7: compute [n]*P
107        # (we keep the output in projective coordinates (X : . : Z) )
108        Q = ladder(n, x0, A, B)
109
110        # step 8-9: from Z we find the prime factor, otherwise we start again

```

```

111     b0, b1 = ZZ(Q[1][0]), ZZ(Q[1][1])
112     multiple_of_p = b0^2 + b1^2*ZZ(H_D[0]) - b0*b1*ZZ(H_D[1])
113     p = gcd(multiple_of_p, n)
114     if p != 0 and p != 1:
115         print(p)
116         break
117
118 solve_challenge()#p=1648645639256828465457544748174133743203704521080088183592
261428006709559740371000545588586782953459910810519785312012157477919148112531
572296425788887357909980069006839772896033556131571761913209774789100699098723
421977321595134129037532546783965002048584341013376047686303874760541756797073
831242819880429174364718452680669209250725839304980026212970039161865245391690
214151258283835742445527838899282031118204275075962879626345800520621185162289
804200127011086044522751809809944249654429148745361686160425429175041188785759
509363655343733117043566047545467214333058929982437607654768905172484478633732
628176922210741735587758783669989674572656762809228299539990596809965524535982
518167134218052215386752045855737810080549588572896551712474911318592407868811
662574389154740724845936118553413765559879484138134507259595217354708891014131
688788908579109860354106618385785504553785122534872843214063342958254735587516
901838714699801065169723631123828130907246257133488650195344127842009139462049
841946194756963053401399250005394289307586303145259312797817386832226850374039
880465334033247875479727266281226123177539562068930094939385942356649231158468
574200680419438734983371309773638965566070108647334976235103297401189088428514
9086019
119
120 p=1648645639256828465457544748174133743203704521080088183592261428006709559740
371000545588586782953459910810519785312012157477919148112531572296425788887357
909980069006839772896033556131571761913209774789100699098723421977321595134129
037532546783965002048584341013376047686303874760541756797073831242819880429174
364718452680669209250725839304980026212970039161865245391690214151258283835742
445527838899282031118204275075962879626345800520621185162289804200127011086044
522751809809944249654429148745361686160425429175041188785759509363655343733117
043566047545467214333058929982437607654768905172484478633732628176922210741735
587758783669989674572656762809228299539990596809965524535982518167134218052215
386752045855737810080549588572896551712474911318592407868811662574389154740724
845936118553413765559879484138134507259595217354708891014131688788908579109860
354106618385785504553785122534872843214063342958254735587516901838714699801065
169723631123828130907246257133488650195344127842009139462049841946194756963053
401399250005394289307586303145259312797817386832226850374039880465334033247875
479727266281226123177539562068930094939385942356649231158468574200680419438734
9833713097736389655660701086473349762351032974011890884285149086019
121 q=n//p
122 assert(p*q==n)
123 phi=(p-1)*(q-1)
124 d=inverse_mod(e,phi)
125 print(long_to_bytes(int(pow(c,d,n))))

```

得到flag为flag{2c8a995547553aa788b43f26a57111edb2588304}

Re

首先我们把程序丢进ida

```

1 int __cdecl __noreturn main_0(int argc, const char **argv, const char **envp)
2 {
3     int v3; // eax
4     char v4[40]; // [esp+78h] [ebp-84h] BYREF
5     char Destination[40]; // [esp+A0h] [ebp-5Ch] BYREF
6     char Str[5]; // [esp+C8h] [ebp-34h] BYREF
7     char Source[13]; // [esp+D0h] [ebp-2Fh] BYREF
8     char v8[22]; // [esp+D0h] [ebp-22h] BYREF
9     char v9[12]; // [esp+F0h] [ebp-Ch] BYREF
10
11     sub_40100A();
12     printf("input your flag:\n");
13     ((void (__cdecl *)(char *))gets)(Str);
14     if ( strlen(Str) == 42 )
15     {
16         sub_40105A(Str);
17         sub_401014();
18         sub_401023((int)Str, 45);
19         sub_401014();
20         strncpy(Destination, Source, 0x20u);
21         Destination[32] = 0;
22         sub_40101E((int)Destination);
23         strncpy(v8, v8, 0x12u);
24         v8[18] = 0;
25         sub_401046(v4);
26         sub_401005(v4);
27         strncpy(v4, v9, 1u);
28         v4[1] = 0;
29         v3 = sub_401019((int)v4);
30         sub_401032(v3);
31         sub_401028();
32     }
33 }

```

从伪代码中我们可以看出,Str也就是flag长度为42,接下来我们一个个分析主函数中出现的函数

首先是sub_40105A

```

1 int __cdecl sub_402B40(char *Source)
2 {
3     int result; // eax
4     int v2; // [esp+4Ch] [ebp-10h]
5     char Destination; // [esp+50h] [ebp-Ch] BYREF
6     char v4; // [esp+51h] [ebp-Bh]
7     char Str1; // [esp+54h] [ebp-8h] BYREF
8     int v6; // [esp+55h] [ebp-7h]
9     char v7; // [esp+59h] [ebp-3h]
10
11     Str1 = 0;
12     v6 = 0;
13     v7 = 0;
14     Destination = 0;
15     v4 = 0;
16     v2 = 13;
17     strncpy(&Str1, Source, 5u);
18     strncpy(&Destination, Source + 41, 10);
19     v7 = 0;
20     v4 = 0;
21     if ( strcmp(&Str1, "flag{") || (result = Destination, Destination != 125) )
22         sub_401028();
23     while ( v2 <= 28 )
24     {
25         if ( !sub_40104B(Source[v2]) )
26             sub_401028();
27         result = v2 + 5;
28         v2 += 5;
29     }
30     return result;
31 }

```

可以看到第一个部分是用于检查flag是否为flag(开头,并以}结束,如果不是,触发函数sub_401028());退出

```

1 void __noreturn sub_40FBE0()
2 {
3     printf("wrong flag!!!\n");
4     if ( --File._cnt < 0 )
5         _filbuf(&File);
6     else
7         ++File._ptr;
8     exit(0);
9 }

```

第二个部分则是检查是否在第13,18,23,28的位置满足某个条件,条件在sub_40104B中

```

1 BOOL __cdecl sub_402B00(char a1)
2 {
3     return a1 == 45;
4 }

```

可以看到这个条件就是第13,18,23,28的位置都是字符"-"

回到主函数,我们继续查看函数sub_401023

```

1 int __cdecl sub_402C40(int a1, char a2)
2 {
3     int result; // eax
4     int v3; // [esp+4Ch] [ebp-8h]
5     int i; // [esp+50h] [ebp-4h]
6
7     v3 = 0;
8     for ( i = 0; ; ++i )
9     {
10         result = *(char *)(i + a1);
11         if ( !*(_BYTE *)(i + a1) )
12             break;
13         if ( *(char *)(i + a1) != a2 )
14         {
15             *(_BYTE *)(v3 + a1) = *(_BYTE *)(i + a1);
16             ++v3;
17         }
18     }
19     *(_BYTE *)(v3 + a1) = 0;
20     return result;
21 }

```

该函数的目的是从输入的字符串 a1 中删除所有等于给定字符 a2的字符。它通过遍历输入字符串，将不等于 a2的字符复制到一个新的字符串中，最后在新字符串的末尾添加字符串结束符。修改后的字符串保存在输入字符串的原址。从传进去的参数我们可以知道这个函数删掉了flag中的四个"-"字符

接下来这一部分将flag中的去掉flag头尾以及字符"-"存进Destination中

```

1 strncpy(Destination, Source, 0x20u);
2 Destination[32] = 0;

```

然后我们再看函数sub_40101E

```
3 | int result; // eax
4 | int i; // [esp+4Ch] [ebp-3Ch]
5 | int v3[13]; // [esp+54h] [ebp-34h]
6 |
7 | v3[0] = 102;
8 | v3[1] = 52;
9 | v3[2] = 51;
10 | v3[3] = 49;
11 | v3[4] = 52;
12 | v3[5] = 57;
13 | v3[6] = 96;
14 | v3[7] = 60;
15 | v3[8] = 61;
16 | v3[9] = 34;
17 | v3[10] = 104;
18 | v3[11] = 33;
19 | v3[12] = 56;
20 | for ( i = 0; i < 13; ++i )
21 | {
22 |     if ( i % 2 )
23 |     {
24 |         if ( (char)((2 * i) ^ *(_BYTE *)(i + a1)) != v3[i] )
25 |             sub_401028();
26 |     }
27 |     else if ( (char)(i ^ *(_BYTE *)(i + a1)) != v3[i] )
28 |     {
29 |         sub_401028();
30 |     }
31 |     result = i + 1;
32 | }
33 | return result;
34 | }
```

可以看到这个函数对前13个字符按奇偶进行了不同的异或操作,我们写代码恢复前13个字符

```
1 | v3 = [102, 52, 51, 49, 52, 57, 96, 60, 61, 34, 104, 33, 56]
2 | a= []
3 | for i in range(13):
4 |     if i % 2:
5 |         a.append(chr(v3[i] ^ (2 * i)))
6 |     else:
7 |         a.append(chr(v3[i] ^ (i)))
8 | print(''.join(a))
```

得到f61703f250b74

```
1 | strncpy(v4, v8, 0x12u);
2 | v4[18] = 0;
```

这一部分将接下来18个字符存进v4

然后我们看函数sub_401046


```
IDA View-A  Pseudocode-B  Pseudocode-A  Hex View-1  Structures  Enums
1 int __cdecl sub_4030E0(char *Str)
2 {
3     int v1; // eax
4     int result; // eax
5     int v3[22]; // [esp+4Ch] [ebp-7Ch] BYREF
6     char v4[16]; // [esp+A4h] [ebp-24h] BYREF
7     char v5[16]; // [esp+B4h] [ebp-14h]
8     int i; // [esp+C4h] [ebp-4h]
9
10    v5[0] = 0x3F;
11    v5[1] = 0xBA;
12    v5[2] = 0xC4;
13    v5[3] = 0x91;
14    v5[4] = 0xC4;
15    v5[5] = 0x74;
16    v5[6] = 2;
17    v5[7] = 0x26;
18    v5[8] = 0xEC;
19    v5[9] = 0x92;
20    v5[10] = 0x38;
21    v5[11] = 0xC2;
22    v5[12] = 0xB;
23    v5[13] = 0x6D;
24    v5[14] = 0x27;
25    v5[15] = 0xD3;
26    sub_40102D((int)v3);
27    v1 = strlen(Str);
28    sub_401041((int)v3, Str, v1);
29    result = sub_40105F((int)v3, (int)v4);
30    for ( i = 0; i < 16; ++i )
31    {
32        if ( v5[i] != v4[i] )
```

先看sub_40102D

```
1 DWORD *__cdecl sub_4010D0(_DWORD *a1)
2 {
3     _DWORD *result; // eax
4
5     *a1 = 0;
6     a1[1] = 0;
7     a1[2] = 0x67452301;
8     result = a1;
9     a1[3] = -271733879;
10    a1[4] = -1732584194;
11    a1[5] = 271733878;
12    return result;
13 }
```

再看看sub_401041

```

1 void *__cdecl sub_401140(_DWORD *a1, char *Src, unsigned int a3)
2 {
3     unsigned int Size; // [esp+4Ch] [ebp-Ch]
4     int v5; // [esp+50h] [ebp-8h]
5     int i; // [esp+54h] [ebp-4h]
6
7     v5 = (*a1 >> 3) & 0x3F;
8     Size = 64 - v5;
9     *a1 += 8 * a3;
10    if ( *a1 < 8 * a3 )
11        ++a1[1];
12    a1[1] += a3 >> 29;
13    if ( a3 < Size )
14    {
15        i = 0;
16    }
17    else
18    {
19        memcpy((char *)a1 + v5 + 24, Src, Size);
20        sub_40100F(a1 + 2, a1 + 6);
21        for ( i = 64 - v5; i + 64 <= a3; i += 64 )
22            sub_40100F(a1 + 2, &Src[i]);
23        v5 = 0;
24    }
25    return memcpy((char *)a1 + v5 + 24, &Src[i], a3 - i);
26 }

```

最后看看sub_40105F

```

1 int __cdecl sub_4012D0(_DWORD *a1, int a2)
2 {
3     int v3; // [esp+4Ch] [ebp-14h]
4     char Src[8]; // [esp+50h] [ebp-10h] BYREF
5     int v5; // [esp+58h] [ebp-8h]
6     unsigned int v6; // [esp+5Ch] [ebp-4h]
7
8     v6 = 0;
9     v5 = 0;
10    v6 = (*a1 >> 3) & 0x3F;
11    if ( v6 >= 0x38 )
12        v3 = 120 - v6;
13    else
14        v3 = 56 - v6;
15    v5 = v3;
16    sub_40103C((int)Src, (int)a1, 8);
17    sub_401041((int)a1, &unk_424A30, v5);
18    sub_401041((int)a1, Src, 8);
19    return sub_40103C(a2, (int)(a1 + 2), 16);
20 }

```

到这里我们已经可以判断sub_401046是用来计算字符的md5值是否为数组v5的结果,但暂时我们还无法找到哈希值的原像,我们再回到主函数看另一个函数sub_401005

```

1 int __cdecl sub_403200(char *Str)
2 {
3     int result; // eax
4     char Str1[24]; // [esp+4Ch] [ebp-98h] BYREF
5     char Str2[128]; // [esp+64h] [ebp-80h] BYREF
6
7     strcpy(Str1, "ZjQ3ODEzYzI2NTk0YzA=");
8     Str[14] = 0;
9     sub_401055(Str, (int)Str2);
10    result = strcmp(Str1, Str2);
11    if ( result )
12        sub_401028();
13    return result;
14 }

```

可以看到这一部分将长度为18的字符截取前14个,并用base64编码,看看是否为ZjQ3ODEzYzI2NTk0YzA=,因此我们可以得到前14个字符为f47813c26594c0

Recipe	Input
<p>From Base64</p> <p>Alphabet A-Za-z0-9+/=</p> <p><input checked="" type="checkbox"/> Remove non-alphabet chars <input type="checkbox"/> Strict mode</p>	ZjQ3ODEzYzI2NTk0YzA=
<p>STEP BAKE! <input checked="" type="checkbox"/> Auto Bake</p>	<p>Output</p> <p>f47813c26594c0</p>

剩下四个字符我们直接利用md5值爆破,得到完整18个字符为f47813c26594c0e581

```

1 from string import digits,ascii_lowercase
2 from itertools import product
3 from hashlib import md5
4
5 lib=digits+ascii_lowercase
6 md5res = '3fbac491c4740226ec9238c20b6d27d3'
7
8 flag2 = 'f47813c26594c0'
9
10 for i in product(lib,repeat=4):
11     i=''.join(i)
12     s=flag2+i
13     h=md5(s.encode()).hexdigest()
14     if h==md5res:
15         print(s)
16         #f47813c26594c0e581
17
18 #flag='flag{f61703f2-50b7-4f47-813c-26594c0e5810}'

```

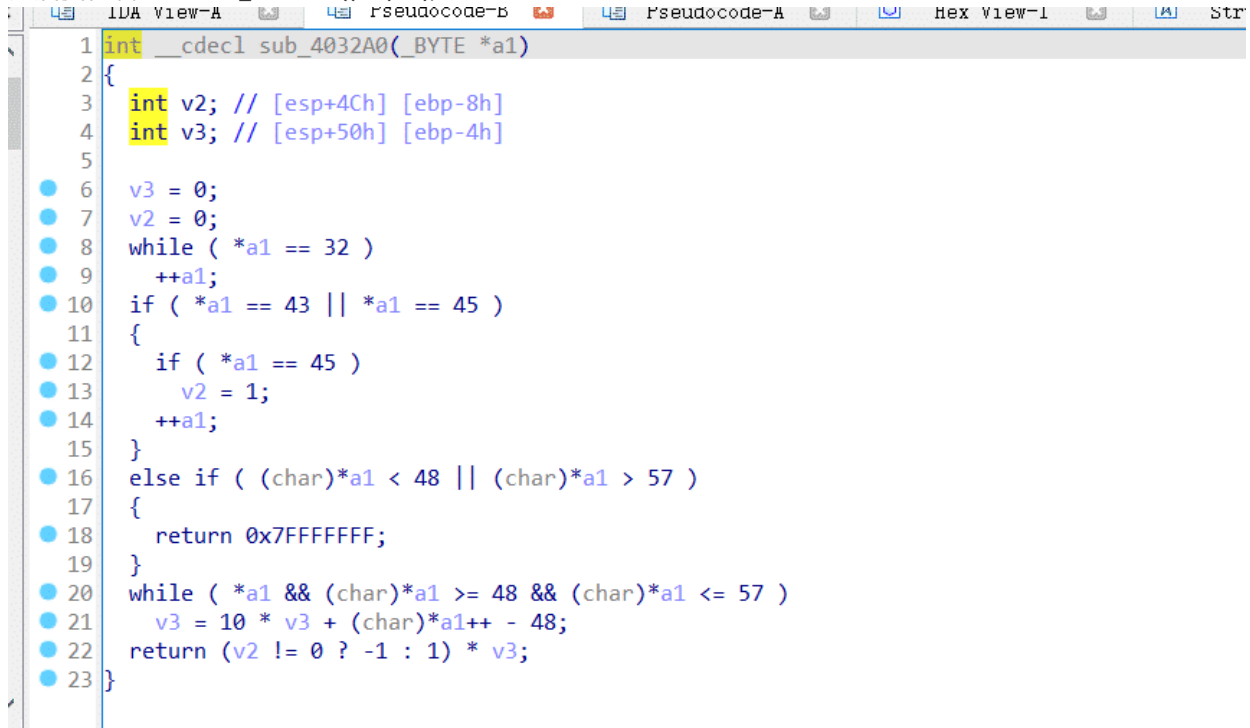
```
19 | #print(len(flag))
```

最后我们只剩下一个未知字符了,回到主函数

```
1 | strncpy(v4, v9, 1u);
2 | v4[1] = 0;
```

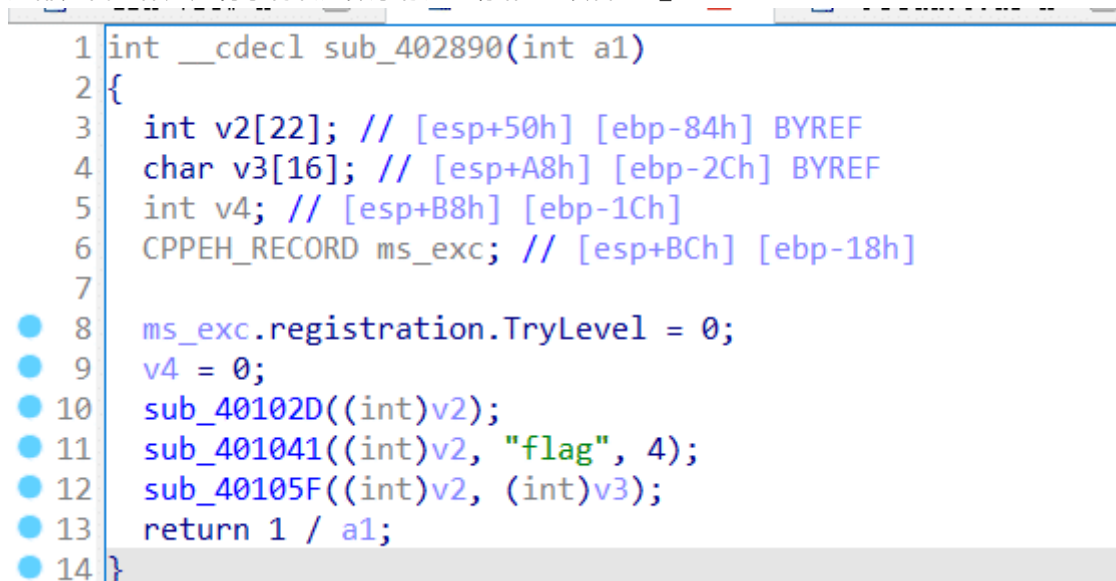
最后一个字符被存进了v4中

然后我们看v3 = sub_401019((int)v4);



```
1 | int __cdecl sub_4032A0(_BYTE *a1)
2 | {
3 |     int v2; // [esp+4Ch] [ebp-8h]
4 |     int v3; // [esp+50h] [ebp-4h]
5 |
6 |     v3 = 0;
7 |     v2 = 0;
8 |     while ( *a1 == 32 )
9 |         ++a1;
10 |    if ( *a1 == 43 || *a1 == 45 )
11 |    {
12 |        if ( *a1 == 45 )
13 |            v2 = 1;
14 |        ++a1;
15 |    }
16 |    else if ( (char)*a1 < 48 || (char)*a1 > 57 )
17 |    {
18 |        return 0xFFFFFFFF;
19 |    }
20 |    while ( *a1 && (char)*a1 >= 48 && (char)*a1 <= 57 )
21 |        v3 = 10 * v3 + (char)*a1++ - 48;
22 |    return (v2 != 0 ? -1 : 1) * v3;
23 | }
```

这部分代码作用是将字符转成数字存进v3,我们继续看sub_401032



```
1 | int __cdecl sub_402890(int a1)
2 | {
3 |     int v2[22]; // [esp+50h] [ebp-84h] BYREF
4 |     char v3[16]; // [esp+A8h] [ebp-2Ch] BYREF
5 |     int v4; // [esp+B8h] [ebp-1Ch]
6 |     CPPEH_RECORD ms_exc; // [esp+BCh] [ebp-18h]
7 |
8 |     ms_exc.registration.TryLevel = 0;
9 |     v4 = 0;
10 |    sub_40102D((int)v2);
11 |    sub_401041((int)v2, "flag", 4);
12 |    sub_40105F((int)v2, (int)v3);
13 |    return 1 / a1;
14 | }
```

这一部分似乎是要我们触发一个异常,而这里由于return 1/a1,因此只要为字符0,传进这个函数中1/0就会产生异常,因此我们的到了最后一个位置字符为0

到这里我们已经知道了全部的字符,我们将其拼起来并加上flag头尾再加上之前删去的字符"-"就得到了
flag{f61703f2-50b7-4f47-813c-26594c0e5810}

