MiniLCTF2023

magical_syacall

考点

SIGNAL反调试、TracePID反调试、双进程、ptrace、Syscall

Ptrace

ptrace系统调用提供了一个进程(`tracer`)可以控制另一个进程(`tracee`)运行的方法，并且`tracer`可以监控和修改`tracee`的内存和寄存器，主要用作实现断点调试和系统调用追踪。

当父进程使用`ptrace`函数并传递`PTRACE_SYSCALL`参数来监控子进程时，它会暂停子进程的执行并等待下一个系统调用。

具体来说，当子进程调用一个系统调用并引发一个中断时，父进程会收到一个`SIGTRAP`信号，并可以通过`waitpid`函数等待子进程进入暂停状态。此时，父进程可以使用`ptrace`函数再次传递`PTRACE_SYSCALL`参数来恢复子进程的执行，并让子进程继续进行系统调用。

在这种情况下，父进程不会修改子进程的寄存器或内存值，只是简单地等待并恢复子进程的执行。这个过程可以反复进行，直到子进程结束或父进程选择终止跟踪操作。

ptrace中的重要数据

```
#define PTRACE_TRACEME 0
#define PTRACE_PEEKTEXT 1
#define PTRACE_PEEKDATA 2
#define PTRACE_PEEKUSR 3
#define PTRACE_POKETEXT 4
#define PTRACE_POKEDATA 5
#define PTRACE_POKEUSR 6
#define PTRACE_CONT 7
#define PTRACE_KILL 8
#define PTRACE_SINGLESTEP 9
#define PTRACE_ATTACH 16
#define PTRACE_DETACH 17
#define PTRACE_SYSCALL 24
#define PTRACE_SETOPTIONS 0x4200
#define PTRACE_GETEVENTMSG 0x4201
#define PTRACE_GETSIGINFO 0x4202
#define PTRACE_SETSIGINFO 0x4203
#define PTRACE_GETREGSET 0x4204
#define PTRACE_SETREGSET 0x4205
```

```
20 #define PTRACE_SEIZE 0x4206
21 #define PTRACE_INTERRUPT 0x4207
22 #define PTRACE_LISTEN 0x4208
23 #define PTRACE_PEEKSIGINFO 0x4209
24
25 #define PTRACE_EVENTMSG_SYSCALL_ENTRY 1
26 #define PTRACE_EVENTMSG_SYSCALL_EXIT 2
27 #define PTRACE_PEEKSIGINFO_SHARED (1 << 0)
28 #define PTRACE_EVENT_FORK 1
29 #define PTRACE_EVENT_VFORK 2
30 #define PTRACE_EVENT_CLONE 3
31 #define PTRACE_EVENT_EXEC 4
32 #define PTRACE_EVENT_VFORK_DONE 5
33 #define PTRACE_EVENT_EXIT 6
34 #define PTRACE_EVENT_SECCOMP 7
35 #define PTRACE_EVENT_STOP 128
36 #define PTRACE_O_TRACESYSGOOD 1
37 #define PTRACE_O_TRACEFORK (1 << PTRACE_EVENT_FORK)
38 #define PTRACE_O_TRACEVFORK (1 << PTRACE_EVENT_VFORK)
39 #define PTRACE_O_TRACECLONE (1 << PTRACE_EVENT_CLONE)
40 #define PTRACE_O_TRACEEXEC (1 << PTRACE_EVENT_EXEC)
41 #define PTRACE_O_TRACEVFORKDONE (1 << PTRACE_EVENT_VFORK_DONE)
42 #define PTRACE_O_TRACEEXIT (1 << PTRACE_EVENT_EXIT)
43 #define PTRACE_O_TRACESECCOMP (1 << PTRACE_EVENT_SECCOMP)
44 #define PTRACE_O_EXITKILL (1 << 20)
45 #define PTRACE_O_SUSPEND_SECCOMP (1 << 21)
46 #define PTRACE_O_MASK (0x000000ff | PTRACE_O_EXITKILL | PTRACE_O_SUSPEND_SECCO
```

在user.h可以得知**user_regs_struct**的结构定义(在不同架构下也不一样，具体如下)，该结构体与**PTRACE_SETREGS**和**PTRACE_GETREGS**有关

user.h

在X86-64下，其结构如下

其中 `orig_rax` 是 `struct user_regs_struct` 结构体中的一个成员变量。它保存了发生系统调用前 `rax` 寄存器中的值，也就是系统调用号。因为在进行系统调用时， `rax` 寄存器被用来传递系统调用号，所以在系统调用处理过程中，内核会将该值保存在 `orig_rax` 寄存器中。

把下面这个结构体导入IDA并修改变量类型

```
struct user_regs_struct {
  unsigned long r15;
  unsigned long r14;
  unsigned long r13;
  unsigned long r12;
  unsigned long rbp;
  unsigned long rbx;
  unsigned long r11;
  unsigned long r10;
  unsigned long r9;
  unsigned long r8;
  unsigned long rax;
  unsigned long rcx;
  unsigned long rdx;
  unsigned long rsi;
  unsigned long rdi;
  unsigned long orig_rax;
```

```
18    unsigned long rip;
19    unsigned long cs;
20    unsigned long eflags;
21    unsigned long rsp;
22    unsigned long ss;
23    unsigned long fs_base;
24    unsigned long gs_base;
25    unsigned long ds;
26    unsigned long es;
27    unsigned long fs;
28    unsigned long gs;
29  };
```

Syscall

常见的系统调用号

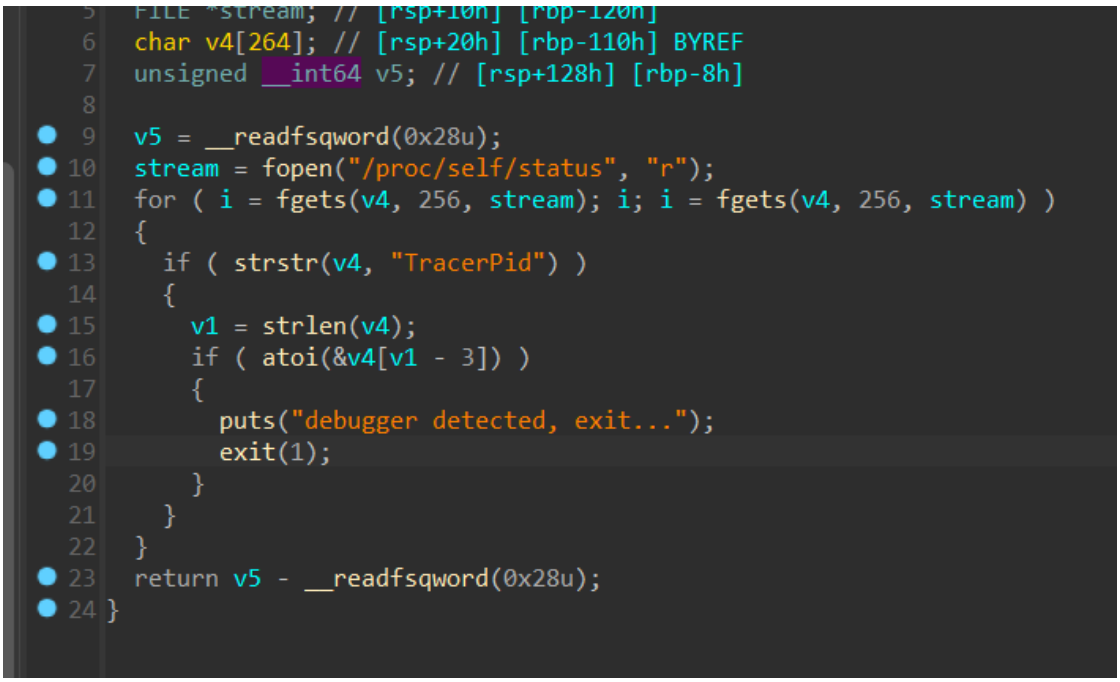
对应Syscall的过程可以参考syscall过程

解题过程

反调试

init_array段存在两处反调试

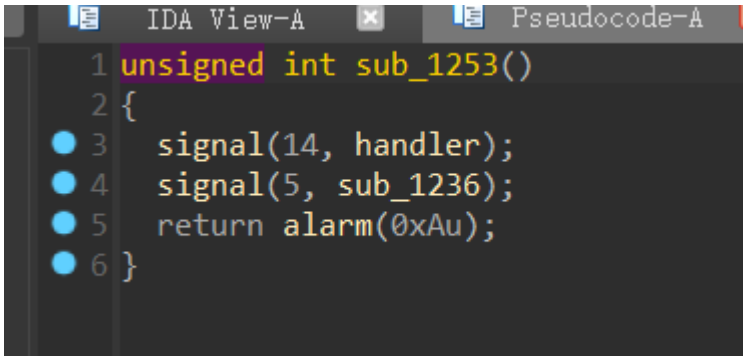
TracerPid的检测



```
FILE *stream; // [rsp+10h] [rbp-120h]
char v4[264]; // [rsp+20h] [rbp-110h] BYREF
unsigned __int64 v5; // [rsp+128h] [rbp-8h]

v5 = __readfsqword(0x28u);
stream = fopen("/proc/self/status", "r");
for ( i = fgets(v4, 256, stream); i; i = fgets(v4, 256, stream) )
{
  if ( strstr(v4, "TracerPid") )
  {
    v1 = strlen(v4);
    if ( atoi(&v4[v1 - 3]) )
    {
      puts("debugger detected, exit...");
      exit(1);
    }
  }
}
return v5 - __readfsqword(0x28u);
}
```

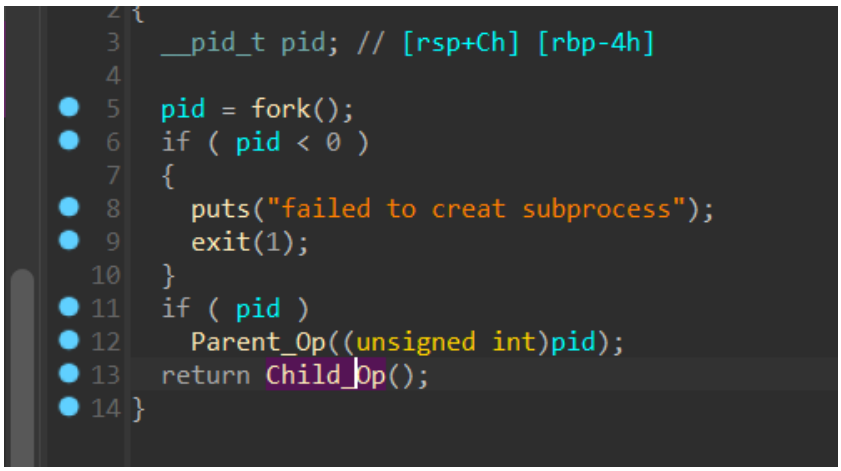
进程创建十秒后自动退出



只需要将call exit语句patch掉即可

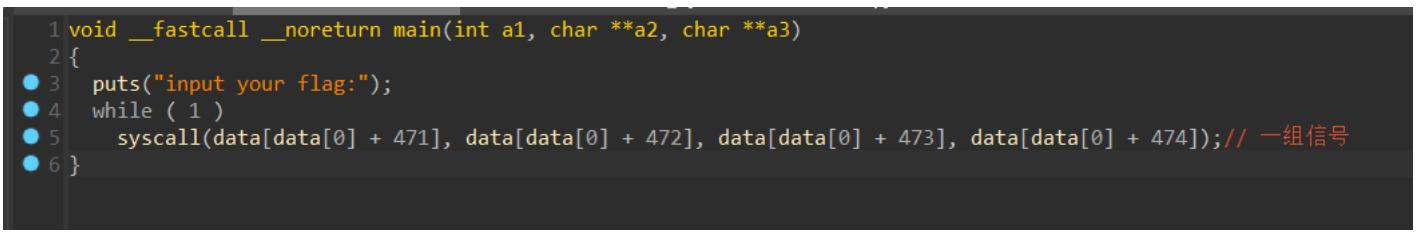
双进程

父进程fork一个子进程，并且父进程通过ptrace来控制子进程



子进程

子进程进行通过while循环进行系统调用，需要关注**data[0]处数据，待会父进程会对该地址的数据进行重写**



父进程

父进程对应的操作如下

```
while ( 1 )
  {
    ptrace(PTRACE_SYSCALL, pid, 0LL, 0LL);
    waitpid(pid, &stat_loc, 0);
    ptrace(PTRACE_GETREGS, pid, 0LL, &REG);
    if ( REG.orig_rax == 0x22B8 )
      break;
```

```
 8      if ( REG.orig_rax == 0x270F )
 9      {
10        puts("congratulations");
11        exit(0);
12      }
13      if ( REG.orig_rax == 0xF3F )
14      {
15        REG.orig_rax = 0LL;
16        REG.rdi = 0LL;
17        REG.rsi = (unsigned __int64)&Data[5];
18        REG.rdx = 1LL;
19        ptrace(PTRACE_SETREGS, pid, 0LL, &REG);
20        ptrace(PTRACE_POKEDATA, pid, Data, (unsigned int)++Data[0]);
21      }
22      if ( REG.orig_rax == 0xF3D )
23      {
24        v2 = 0LL;
25        v3 = 0LL;
26        v4 = 0LL;
27        if ( REG.rdi )
28        {
29          if ( REG.rdi == 1 )
30            v2 = &Data[3];
31        }
32        else
33        {
34          v2 = &Data[2];
35        }
36        if ( REG.rsi )
37        {
38          if ( REG.rsi == 1 )
39          {
40            v3 = Key;
41          }
42          else if ( REG.rsi == 2 )
43          {
44            v3 = Enc;
45          }
46        }
47        else
48        {
49          v3 = dword_41BC;
50        }
51        if ( REG.rdx )
52        {
53          if ( REG.rdx == 1 )
54          {
```

```
55            v4 = &Data[3];
56          }
57          else if ( REG.rdx == 2 )
58          {
59            v4 = &Data[4];
60          }
61        }
62        else
63        {
64          v4 = &Data[2];
65        }
66        *v2 += v3[*v4];
67        Data[0] += 4;
68        ptrace(PTRACE_POKEDATA, pid, Data, (unsigned int)Data[0]);
69      }
70      if ( REG.orig_rax == 0xF3E )
71      {
72        Addr = 0LL;
73        if ( REG.rdi )
74        {
75          if ( REG.rdi == 1 )
76            Addr = &Data[3];
77        }
78        else
79        {
80          Addr = &Data[2];
81        }
82        *Addr = (unsigned int)*Addr % REG.rsi;
83        Data[0] += 3;
84        ptrace(PTRACE_POKEDATA, pid, Data, (unsigned int)Data[0]);
85      }
86      if ( REG.orig_rax == 0xF40 )
87      {
88        Data[5] = ptrace(PTRACE_PEEKDATA, pid, &Data[5], 0LL);
89        Data[++Data[1] + 7] = Data[5];
90        ptrace(PTRACE_POKEDATA, pid, Data, (unsigned int)++Data[0]);
91      }
92      if ( REG.orig_rax == 0xF41 )
93      {
94        Data[5] = Data[Data[1] + 7];
95        --Data[1];
96        ptrace(PTRACE_POKEDATA, pid, Data, (unsigned int)++Data[0]);
97      }
98      if ( REG.orig_rax == 0xF42 )
99      {
100       if ( REG.rdi )
101       {
```

```
102         if ( REG.rdi == 1 )
103           Data[6] = Data[2] == Data[5];
104       }
105       else
106       {
107         Data[6] = Data[4] == REG.rsi;
108       }
109       Data[0] += 3;
110       ptrace(PTRACE_POKEDATA, pid, Data, (unsigned int)Data[0]);
111     }
112     if ( REG.orig_rax == 0xF43 )
113     {
114       if ( Data[6] )
115       {
116         Data[0] = REG.rdi;
117         ptrace(PTRACE_POKEDATA, pid, Data, LODWORD(REG.rdi));
118       }
119       else
120       {
121         Data[0] += 2;
122         ptrace(PTRACE_POKEDATA, pid, Data, (unsigned int)Data[0]);
123       }
124     }
125     if ( REG.orig_rax == 0xF44 )
126     {
127       if ( Data[6] )
128       {
129         Data[0] += 2;
130         ptrace(PTRACE_POKEDATA, pid, Data, (unsigned int)Data[0]);
131       }
132       else
133       {
134         Data[0] = REG.rdi;
135         ptrace(PTRACE_POKEDATA, pid, Data, LODWORD(REG.rdi));
136       }
137     }
138     if ( REG.orig_rax == 0xF45 )
139     {
140       if ( REG.rdi )
141       {
142         switch ( REG.rdi )
143         {
144           case 1uLL:
145             dword_41BC[Data[4]] = Data[4];
146             break;
147           case 2uLL:
148             Data[2] = dword_41BC[Data[4]];
```

```
149             break;
150           case 3uLL:
151             Data[2] = dword_41BC[Data[4] + 267];
152             break;
153           case 4uLL:
154             Data[2] = dword_41BC[Data[2]];
155             break;
156         }
157       }
158       else
159       {
160         Data[2] = Data[4];
161       }
162       Data[0] += 2;
163       ptrace(PTRACE_POKEDATA, pid, Data, (unsigned int)Data[0]);
164     }
165     if ( REG.orig_rax == 0xF46 )
166     {
167       Data[5] ^= Data[2];
168       ptrace(PTRACE_POKEDATA, pid, Data, (unsigned int)++Data[0]);
169     }
170     if ( REG.orig_rax == 0xF47 )
171     {
172       if ( REG.rdi )
173       {
174         if ( REG.rdi == 1 )
175           v6 = &Data[3];
176       }
177       else
178       {
179         v6 = &Data[4];
180       }
181       ++*v6;
182       Data[0] += 2;
183       ptrace(PTRACE_POKEDATA, pid, Data, (unsigned int)Data[0]);
184     }
185     if ( REG.orig_rax == 0xF48 )
186     {
187       dword_41BC[Data[4]] = dword_41BC[Data[3]];
188       dword_41BC[Data[3]] = dword_41BC[Data[4]];
189       ptrace(PTRACE_POKEDATA, pid, Data, (unsigned int)++Data[0]);
190     }
191     if ( REG.orig_rax == 0xF49 )
192     {
193       v7 = 0LL;
194       if ( REG.rdi )
195       {
```

```
196         if ( REG.rdi == 1 )
197           v7 = &Data[3];
198       }
199       else
200       {
201         v7 = &Data[4];
202       }
203       *v7 = 0;
204       Data[0] += 2;
205       ptrace(PTRACE_POKEDATA, pid, Data, (unsigned int)Data[0]);
206     }
207     ptrace(PTRACE_SYSCALL, pid, 0LL, 0LL);
208     waitpid(pid, &stat_loc, 0);
```

上述代码存在两处wait操作，其中**第一处是为了等待子进程进行系统调用，在子进程进行系统调用前将信号传给父进程；第二处是为了让子进程继续往下执行系统调用，父进程进行等待。syscall主要是通过rsi、rdi、rdx寄存器进行传参，在本题中机器码后跟着的数据就是参数，会传入对应的寄存器中**

```
1 ptrace(PTRACE_SYSCALL, pid, 0LL, 0LL);
2 waitpid(pid, &stat_loc, 0);
```

下面的语句是将Data[0]处的数据，写入子进程中的Data地址，由于父进程fork时，子进程会复制父进程的地址空间，包括变量的存储地址。因此，子进程中同一变量的存储地址和父进程中的地址是一样的，所以这相当于重写了子进程中Data[0]的数据，进而**修改子进程系统调用号，相当于EIP**

```
1 ptrace(PTRACE_POKEDATA, pid, Data, (unsigned int)Data[0]);
```

通过`ptrace(PTRACE_GETREGS, pid, 0LL, &REG);`获取到子进程系统调用前的寄存器即可获取系统调用号，并在之后根据系统调用号来控制操作，每个系统调用号就相当于一个opcode

VM

这一部分就是理解每个Opcode对应的操作了，这里我是动态调试的(**更好的办法应该是解析每个Opcode的含义然后写脚本转为代码**)，通过关键的一些操作知道对数据进行魔改RC4加密，进而解密出Flag

下面是一些细节

读取输入数据/key/密文

```
47      if ( REG.rdi )
48      {
49        if ( REG.rdi == 1 )
50          v2 = &Data[3];
51      }
52      else
53      {
54        v2 = &Data[2];
55      }
56      if ( REG.rsi )
57      {
58        if ( REG.rsi == 1 )
59        {
60          v3 = Key;
61        }
62        else if ( REG.rsi == 2 )
63        {
64          v3 = Enc;
65        }
66      }
67      else
68      {
69        v3 = dword_41BC;
70      }
71      if ( REG.rdx )
72      {
73        if ( REG.rdx == 1 )
74        {
75          v4 = &Data[3];
76        }
77        else if ( REG.rdx == 2 )
78        {
79          v4 = &Data[4];
80        }
81      }
82      else
83      {
84        v4 = &Data[2];
85      }
86      *v2 += v3[*v4];
87      Data[0] += 4;
88      ptrace(PTRACE_POKEDATA, pid, Data, (unsigned int)Data[0
89    }
```

异或

```
if ( REG.orig_rax == 0xF46 )
{
  Data[5] ^= Data[2];
  ptrace(PTRACE_POKEDATA, pid, Data, (unsigned int)++Data[0]);
}
if ( REG.orig_rax == 0xF47 )
```

被修改的Swap(RC4魔改的地方)

```
if ( REG.orig_rax == 0xF48 )
{
  dword_41BC[Data[4]] = dword_41BC[Data[3]];
  dword_41BC[Data[3]] = dword_41BC[Data[4]];
  ptrace(PTRACE_POKEDATA, pid, Data, (unsigned int)++Data[0]);
}
```

比较

```
if ( REG.orig_rax == 0xF42 )
{
  if ( REG.rdi )
  {
    if ( REG.rdi == 1 )
      Data[6] = Data[2] == Data[5];
  }
  else
  {
    Data[6] = Data[4] == REG.rsi;
  }
  Data[0] += 3;
  ptrace(PTRACE_POKEDATA, pid, Data, (unsigned int)Data[0]);
}
```

取余操作(余数为系统操作号的后面第两位)

```
if ( REG.orig_rax == 0xF3E )
{
  Addr = 0LL;
  if ( REG.rdi )
  {
    if ( REG.rdi == 1 )
      Addr = &Data[3];
  }
  else
  {
    Addr = &Data[2];
  }
  *Addr = (unsigned int)*Addr % REG.rsi;
  Data[0] += 3;
  ptrace(PTRACE_POKEDATA, pid, Data, (unsigned int)Data[0]);
}
```

通过对Opcode的分析已经能大致理解出来是RC4(**%256，%key_len**)

```
    0x00000002,

    0x00000F49, 0x00000000, // 获取循环的长度
    0x00000F45, 0x00000001,
    0x00000F47, 0x00000000, 0x00000F42, 0x00000000, 0x00000100, 0x00000F44, 0x0000000D,

    0x00000F49, 0x00000000,
    0x00000F49, 0x00000001, 0x00000F3D, 0x00000001, 0x00000000, 0x00000002, 0x00000F45, 0x00000000,
    0x00000F3E, 0x00000000, 0x0000000C, 0x00000F3D, 0x00000001, 0x00000001, 0x00000000, 0x00000F3E,
    0x00000001, 0x00000100, 0x00000F48, 0x00000F47, 0x00000000, 0x00000F42, 0x00000000, 0x00000100,
    0x00000F44, 0x0000001A,

    0x00000F49, 0x00000000,
    0x00000F49, 0x00000001, 0x00000F47, 0x00000000,
    0x00000F3D, 0x00000001, 0x00000000, 0x00000002, 0x00000F3E, 0x00000001, 0x00000100, 0x00000F48,
    0x00000F45, 0x00000002, 0x00000F3D, 0x00000000, 0x00000000, 0x00000001, 0x00000F3E, 0x00000000,
    0x00000100, 0x00000F45, 0x00000004, 0x00000F41, 0x00000F46, 0x00000F45, 0x00000003, 0x00000F42,
    0x00000001, 0x00000000, 0x00000F44, 0x0000005A, 0x00000F42, 0x00000000, 0x00000020, 0x00000F44,
    0x00000036, 0x0000270F, 0x000022B8, 0x00000000};
```

解题脚本

对魔改RC4进行解密(**其实也可以直接在0xF42下断点，之后获取密钥流和密文直接异或即可**)

```
#include<stdio.h>
#include <string.h>
#include<Windows.h>


void Rc4_init(unsigned char* S, unsigned char* K, unsigned char* key, unsigned
{
        unsigned char  tmp = 0;
        for (long long i = 0; i < 256; ++i)
        {
                S[i] = i;
                K[i] = key[i % len];
        }
        int j = 0;
        for (int i = 0; i < 256; ++i)
        {
                j = (j + S[i] + K[i]) % 256;
                S[i] = S[j];
                S[j] = S[i];
        }

        return;
}
void Rc4_encrypt(unsigned char* S, unsigned  char* flag, int len)
{

        int i = 0, j = 0, t = 0;
        unsigned char tmp = 0;
```

```
        for (unsigned long long k = 0; k < len; ++k)
        {
                i = (i + 1) % 256;
                j = (j + S[i]) % 256;
                S[i] = S[j];
                S[j] = S[i];
                t = (S[i] + S[j]) % 256;
                flag[k] ^= S[t];

        }
}
int main()
{

        unsigned char S[256] = { 0 };
        unsigned char K[256] = { 0 };
        char code[] = { 0x00000093, 0x000000A3, 0x000000CB, 0x000000C9, 0x0000
        0x000000B1, 0x0000001A, 0x00000054, 0x0000009B, 0x00000050, 0x000000CB
        0x000000EB, 0x0000000F, 0x000000B2, 0x0000008D, 0x0000002F, 0x000000E6
        0x000000B5, 0x0000003D, 0x000000D7, 0x0000009C, 0x000000C5, 0x00000081
        0x00000090, 0x000000F1, 0x0000009B, 0x000000AB, 0x0000002F, 0x000000F2
        char decode[100] = { 0 };
        char key[] = "MiniLCTF2023";
        Rc4_init(S, K, (unsigned char*)key, strlen(key));
        Rc4_encrypt(S, (unsigned char*)code, 38);
                for (int i = 37; i >= 0; --i) {
                printf("%c", code[i]);
        }
        return 0;
}
```

getflag:a_v1rtu@l_m@ch1ne_w1th_ma9ical_sy$call

VM解释器

这里是我解出flag后写的解释器，感觉写的不太好，完整地打印了加密地过程，密钥流也可以在里面提取出来(因为我patch了比较语句为永真)

```
#include<stdio.h>
#include<string.h>

unsigned int enc[] = { 0x00000093, 0x000000A3, 0x000000CB, 0x000000C9, 0x00000
                0x000000B1, 0x0000001A, 0x00000054, 0x0000009B, 0x00000050, 0x
                0x000000EB, 0x0000000F, 0x000000B2, 0x0000008D, 0x0000002F, 0x
                0x000000B5, 0x0000003D, 0x000000D7, 0x0000009C, 0x000000C5, 0x
                0x00000090, 0x000000F1, 0x0000009B, 0x000000AB, 0x0000002F, 0x
```

```
 9
10 unsigned int Key[12] = {
11     0x0000004D, 0x00000069, 0x0000006E, 0x00000069, 0x0000004C, 0x00000043, 0x
12     0x00000032, 0x00000030, 0x00000032, 0x00000033
13 };
14 unsigned int Data[100] = {
15     0x00000000, 0xFFFFFFFF, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x
16 };
17
18 unsigned int S_Table[300] = { 0 };
19
20 unsigned int opcode[92] = {
21     0x00000F49, 0x00000000,
22     0x00000F3F, 0x00000F40,
23     0x00000F47, 0x00000000,
24     0x00000F42, 0x00000000, 0x00000026, // 可能是长度,for循环 read?
25     0x00000F44,                         // continue?
26     0x00000002,
27
28     0x00000F49, 0x00000000, // 获取循环的长度
29     0x00000F45, 0x00000001,
30     0x00000F47, 0x00000000, 0x00000F42, 0x00000000, 0x00000100, 0x00000F44, 0x
31
32     0x00000F49, 0x00000000,
33     0x00000F49, 0x00000001, 0x00000F3D, 0x00000001, 0x00000000, 0x00000002, 0x
34     0x00000F3E, 0x00000000, 0x0000000C, 0x00000F3D, 0x00000001, 0x00000001, 0x
35     0x00000001, 0x00000100, 0x00000F48, 0x00000F47, 0x00000000, 0x00000F42, 0x
36     0x00000F44, 0x0000001A,
37
38     0x00000F49, 0x00000000,
39     0x00000F49, 0x00000001, 0x00000F47, 0x00000000,
40     0x00000F3D, 0x00000001, 0x00000000, 0x00000002, 0x00000F3E, 0x00000001, 0x
41     0x00000F45, 0x00000002, 0x00000F3D, 0x00000000, 0x00000000, 0x00000001, 0x
42     0x00000100, 0x00000F45, 0x00000004, 0x00000F41, 0x00000F46, 0x00000F45, 0x
43     0x00000001, 0x00000000, 0x00000F44, 0x0000005A, 0x00000F42, 0x00000000, 0x
44     0x00000036, 0x0000270F, 0x000022B8, 0x00000000 };
45
46 char input[] = "12345678901234567890123456789012345678";
47 int main() {
48     int i = 0;
49     int com = 0;
50     unsigned int* dest = 0;
51     unsigned int* index = 0;
52     unsigned int* source = 0;
53     while (1) {
54         switch (opcode[i]) {
55         case 0xF49:
```

```
56              if (opcode[i + 1]) {
57                  if (opcode[i + 1] == 1) {
58                      Data[3] = 0;
59                      printf("Data[3] <= 0\n");
60                  }
61              }
62              else {
63                  Data[4] = 0;
64                  printf("Data[4] <= 0\n");
65              }
66              i += 2;
67              break;
68          case 0xF3F:
69              Data[5] = input[Data[4]];
70              printf("Read Data: Data[5] <= %d\n", input[Data[4]]);
71              i += 1;
72              break;
73          case 0xF40:
74              Data[++Data[1] + 7] = Data[5];
75              printf("Data[%d] <= Data[5]\n", Data[1] + 7);
76              i += 1;
77              break;
78          case 0xF47:
79              if (opcode[i + 1]) {
80                  if (opcode[i + 1] == 1) {
81                      Data[3] += 1;
82                      printf("Data[3] += 1\n");
83                  }
84              }
85              else {
86                  Data[4] += 1;
87                  printf("Data[4] += 1\n");
88              }
89              i += 2;
90              break;
91          case 0xF42:
92              if (opcode[i + 1]) {
93                  if (opcode[i + 1] == 1) {
94                      //Data[6] = Data[2] != Data[5];
95                      Data[6] = 1;
96
97                      printf("Data[6] <= 0x%X\n", 1);
98                  }
99              }
100             else {
101                 Data[6] = opcode[i + 2] == Data[4];
102                 printf("Data[6] <= 0x%X\n", opcode[i + 2] == Data[4]);
```

```
            }
            i += 3;
            break;
        case 0xF44:
            if (Data[6]) {
                i += 2;
            }else{
                i = opcode[i + 1];
            }
            break;
        case 0xF45:
            if (opcode[i + 1]) {
                switch (opcode[i + 1]) {
                case 1:
                    S_Table[Data[4]] = Data[4];
                    printf("S_Table[%d] <= 0x%X\n", Data[4], Data[4]);
                    break;
                case 2:
                    Data[2] = S_Table[Data[4]];
                    printf("Data[2] <= 0x%X\n", S_Table[Data[4]]);
                    break;
                case 3:
                    Data[2] = enc[Data[4] - 1];
                    printf("Data[2] <= 0x%X\n", enc[Data[4] - 1]);
                    break;
                case 4:
                    Data[2] = S_Table[Data[2]];
                    printf("Data[2] <= 0x%X\n", S_Table[Data[2]]);
                    break;
                }
            }
            else {
                Data[2] = Data[4];
                printf("Data[2] <= Data[4]\n");
            }
            i += 2;
            break;
        case 0xF3D:
            if (opcode[i + 1]) {
                if (opcode[i + 1]) {
                    dest = &Data[3];
                    printf("Data[3] += ");
                }
            }
            else {
                dest = &Data[2];
                printf("Data[2] += ");
```

```
150
151             }
152             if (opcode[i + 2]) {
153                 if (opcode[i + 2] == 1) {
154                     source = Key;
155                     printf("Key[");
156                 }
157                 else if (opcode[i + 2] == 2) {
158                     source = enc;
159                     printf("Enc[");
160
161                 }
162             }
163             else {
164                 source = S_Table;
165                 printf("S_Table[");
166             }
167             if (opcode[i + 3]) {
168                 if (opcode[i + 3] == 1) {
169                     index = &Data[3];
170                     printf("%d]\n", Data[3]);
171                 }
172                 else if (opcode[i + 3] == 2) {
173                     index = &Data[4];
174                     printf("%d]\n", Data[4]);
175                 }
176             }
177             else {
178                 index = &Data[2];
179                 printf("%d]\n", Data[2]);
180             }
181             *dest += source[*index];
182             i += 4;
183             break;
184         case 0xF3E:
185             if (opcode[i + 1]) {
186                 if (opcode[i + 1] == 1) {
187                     Data[3] %= opcode[i + 2];
188                     printf("Data[3] %%= 0x%X\n", opcode[i + 2]);
189                 }
190             }
191             else {
192                 Data[2] %= opcode[i + 2];
193                 printf("Data[2] %%= 0x%X\n", opcode[i + 2]);
194             }
195             i += 3;
196             break;
```

```
197         case 0xF48:
198             //SWap:
199             S_Table[Data[4]] = S_Table[Data[3]];
200             S_Table[Data[3]] = S_Table[Data[4]];
201             printf("S_Table[%d] = S_Table[%d]\n", Data[4], Data[3]);
202             printf("S_Table[%d] = S_Table[%d]\n", Data[3], Data[4]);
203             i += 1;
204             break;
205         case 0xF41:
206             Data[5] = Data[Data[1] + 7];//input
207             printf("Data[5] <= 0x%X\n", Data[Data[1] + 7]);
208             --Data[1];
209             i += 1;
210             break;
211         case 0xF46:
212             Data[5] ^= Data[2];
213             printf("Data[5] ^= 0x%X\n", Data[2]);
214             i += 1;
215             break;
216         case 0x0000270F:
217             printf("Right\n");
218             com = 1;
219             i += 2;
220             break;
221         case 0x000022B8:
222             printf("Wrong\n");
223             i += 1;
224             break;
225         default:
226             break;
227         }
228         if (com == 1) {
229             break;
230         }
231     }
232         return 0;
233 }
```

Easy Pass

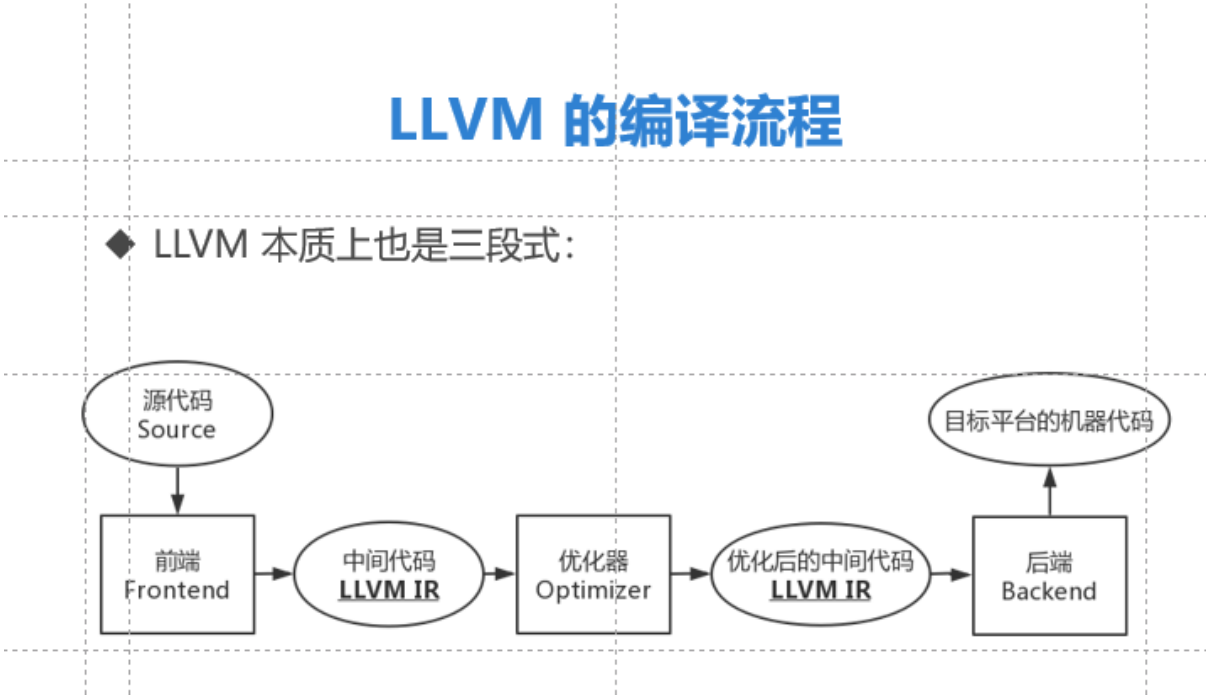
考点

LLVM Pass、VM

解题过程

LLVM项目是一个模块化的、可重用的编译器和工具链集合。

其编译流程如下



通过opt可以加载LLVM Pass可以对中间代码进行优化，本题附件有一个LLVM Pass、中间代码的机器码文件(.bc)以及一个使用文件。

LLVM PASS PWN(一) - 先知社区介绍了如何逆向一个LLVM Pass

LLVM PASS PWN-安全客 - 安全资讯平台介绍了如何通过IDA来调试LLVM Pass

本题中给出的LLVM Pass是FunctionPass，也就是说该Pass会遍历函数并对其做处理

所以我先把main.bc通过clang编译为ELF文件

反编译结果如下

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  aaCCzCzzMMaMCMa();
  bbMMyMyyZZbZMZb();
  ccCCxCxxJJcJCJc(argc, argv);
  ddQQwQwwMMdMQMd();
  eeYYvYvvKKeKYKe();
  ffHHuHuuCCfCHCf();
  ggDDtDttKKgKDKg();
  hhDDsDssOOhODOh();
  iiCCrCrrIIiICIi();
  jjOOqOqqDDjDODj();
  kkSSpSppEEkESEk();
  llXXoXooTTlTXTl();
  mmLLnLnnVVmVLVm();
  nnII_I__HHnHIHn();
  ooOO_O__CCoCOCo();
  ppFF_F__NNpNFNp();
  qqDD_D__BBqBDBq();
  rrTT_T__BBrBTBr();
  ssJJ_J__CCsCJCs();
  ttGG_G__XXtXGXt();
  uuDD_D__OOuODOu();
  vvUU_U__JJvJUJv();
  wwOO_O__SSwSOSw();
  xxRR_R__SSxSRSx();
  yyEE_E__KKyKEKy();
  zzJJ_J__TTzTJTz();
  printf("%s", s);
  return 0;
```

定位runOnFunction

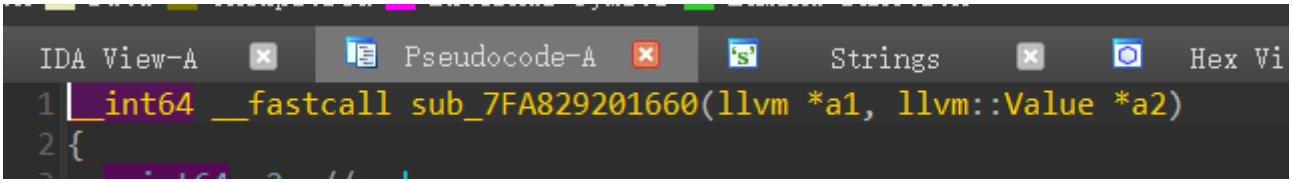
之后通过上述方法定位到runOnFunction方法(该方法被去除符号了不能直接通过函数名查找，但是我们可以自己编写一个LLVM Pass然后通过函数名定位到runOnFunction方法，通过交叉引用也能找到虚表所在的段，如下)

```
.data.rel.ro:0000000000015D43 00                      db    0
.data.rel.ro:0000000000015D44 00                      db    0
.data.rel.ro:0000000000015D45 00                      db    0
.data.rel.ro:0000000000015D46 00                      db    0
.data.rel.ro:0000000000015D47 00                      db    0
.data.rel.ro:0000000000015D48 14 D6 00 00 00 00 00 00 off_15D48 dq offset _ZN12_GLOBAL__N_16MyPassD2Ev
.data.rel.ro:0000000000015D48                                                 ; DATA XREF: `anonymous namespace'::MyPass::My
.data.rel.ro:0000000000015D48                                                 ; `anonymous namespace'::MyPass::~MyPass()+101
.data.rel.ro:0000000000015D48                                                 ; `anonymous namespace'::MyPass::~MyPass()
.data.rel.ro:0000000000015D50 42 D6 00 00 00 00 00 00       dq offset _ZN12_GLOBAL__N_16MyPassD0Ev  ; `anonymous namespace'::MyPass::~MyPass()
.data.rel.ro:0000000000015D58 38 6A 01 00 00 00 00 00       dq offset _ZNK4llvm4Pass11getPassNameEv ; llvm::Pass::getPassName(void)
.data.rel.ro:0000000000015D60 14 E4 00 00 00 00 00 00       dq offset _ZN4llvm4Pass16doInitializationERNS_6ModuleE ; llvm::Pass::doInitialization(
.data.rel.ro:0000000000015D68 2C E4 00 00 00 00 00 00       dq offset _ZN4llvm4Pass14doFinalizationERNS_6ModuleE ; llvm::Pass::doFinalization(llvm
.data.rel.ro:0000000000015D70 B8 69 01 00 00 00 00 00       dq offset _ZNK4llvm4Pass5printERNS_11raw_ostreamEPKNS_6ModuleE ; llvm::Pass::print(llv
.data.rel.ro:0000000000015D78 58 6A 01 00 00 00 00 00       dq offset _ZNK4llvm12FunctionPass17createPrinterPassERNS_11raw_ostreamERKNSt7__cxx111
.data.rel.ro:0000000000015D80 40 6A 01 00 00 00 00 00       dq offset _ZN4llvm12FunctionPass17assignPassManagerERNS_7PMStackENS_15PassManagerTypeE
.data.rel.ro:0000000000015D88 08 6A 01 00 00 00 00 00       dq offset _ZN4llvm4Pass18preparePassManagerERNS_7PMStackE ; llvm::Pass::preparePassMan
.data.rel.ro:0000000000015D90 F8 69 01 00 00 00 00 00       dq offset _ZNK4llvm12FunctionPass27getPotentialPassManagerTypeEv ; llvm::FunctionPass:
.data.rel.ro:0000000000015D98 30 6A 01 00 00 00 00 00       dq offset _ZNK4llvm4Pass16getAnalysisUsageERNS_13AnalysisUsageE ; llvm::Pass::getAnaly
.data.rel.ro:0000000000015DA0 48 6A 01 00 00 00 00 00       dq offset _ZN4llvm4Pass13releaseMemoryEv ; llvm::Pass::releaseMemory(void)
.data.rel.ro:0000000000015DA8 D8 69 01 00 00 00 00 00       dq offset _ZN4llvm4Pass26getAdjustedAnalysisPointerEPKv ; llvm::Pass::getAdjustedAnaly
.data.rel.ro:0000000000015DB0 90 69 01 00 00 00 00 00       dq offset _ZN4llvm4Pass18getAsImmutablePassEv ; llvm::Pass::getAsImmutablePass(void)
.data.rel.ro:0000000000015DB8 10 6A 01 00 00 00 00 00       dq offset _ZN4llvm4Pass18getAsPMDataManagerEv ; llvm::Pass::getAsPMDataManager(void)
.data.rel.ro:0000000000015DC0 18 6A 01 00 00 00 00 00       dq offset _ZNK4llvm4Pass14verifyAnalysisEv ; llvm::Pass::verifyAnalysis(void)
.data.rel.ro:0000000000015DC8 D0 69 01 00 00 00 00 00       dq offset _ZN4llvm4Pass17dumpPassStructureEj ; llvm::Pass::dumpPassStructure(uint)
.data.rel.ro:0000000000015DD0 76 D2 00 00 00 00 00 00       dq offset _ZN12_GLOBAL__N_16MyPass13runOnFunctionERN4llvm8FunctionE ; `anonymous names
.data.rel.ro:0000000000015DD0                               _data_rel_ro ends
```

定位到runOnFunction之后就可以开始逆向了

runOnFunction逆向分析

首先看一下函数参数，这里的a1不知道是什么，但是a2是llvm中的Value对象，在llvm中Function是Value的一个子类



所以上图中的a2应该对应下图中F参数

```
  virtual bool runOnFunction(Function &F) {
    Module *m = F.getParent();
    outs() << m->getTargetTriple() << "\n";
    return true;
  }
};
```

首先通过**getName**获取函数名称，转为C语言字符串之后存入data数组中

```
1017  for ( i = 0; i < 15; ++i )
1018  {
1019    v11[0] = llvm::Value::getName(value);
1020    v11[1] = v2;
1021    llvm::StringRef::str[abi:cxx11](v12, v11);
1022    v3 = (_BYTE *)std::string::c_str(v12);
1023    data[i + 1] = v3[i];
1024    std::string::~string(v12);
1025  }
```

之后通过和d3sky一样的虚拟机来获取存储于data数组尾部的字符串进行加密(题目要求我们patch该处的字符串进行验证，也就相当于输入)

```
unsigned __int64 sub_7FA82920DB60()
{
  unsigned __int64 result; // rax
  int i; // [rsp+0h] [rbp-1Ch]
  unsigned __int8 e3; // [rsp+7h] [rbp-15h]
  unsigned __int8 e1; // [rsp+9h] [rbp-13h]
  unsigned __int8 e2; // [rsp+Ah] [rbp-12h]

  result = (unsigned __int64)data;
  data[0] = 1;
  for ( i = 0; i < 5; ++i )
  {
    e2 = data[data[0] + 1];
    e1 = data[data[0]];
    e3 = data[data[0] + 2];
    data[0] += 3;
    data[e3] = ~(data[e2] & data[e1]);
    result = (unsigned int)(i + 1);
  }
  return result;
}
```

当该FunctionPass遍历了26个函数之后(**通过调试会发现遍历的函数就是之前贴出的那26个函数**)开始进行密文比较。经过26轮加密后进行密文比较

```
1030  strcpy(v9, "HnH59iuc9mk`9x~xpwg");
1031  sub_7FA82920DB60();
1032  if ( ++l == 26 )
1033  {
1034    if ( (Compare((__int64)a1) & 1) != 0 )
1035    {
1036      for ( j = 0; j < 31; ++j )
1037      {
1038        *((_BYTE *)v10 + j) ^= 0x19u;
1039        a1 = (llvm *)llvm::outs(a1);
1040        llvm::raw_ostream::operator<<(a1, *((unsigned __int8 *)v10 + j));
1041      }
1042      v4 = llvm::outs(a1);
1043      llvm::raw_ostream::operator<<(v4, "\n");
1044    }
1045    else
1046    {
1047      for ( k = 0; k < 19; ++k )
1048      {
1049        v9[k] ^= 0x19u;
1050        a1 = (llvm *)llvm::outs(a1);
1051        llvm::raw_ostream::operator<<(a1, (unsigned __int8)v9[k]);
1052      }
1053      v5 = llvm::outs(a1);
1054      llvm::raw_ostream::operator<<(v5, "\n");
1055    }
1056  }
```

解密

该Pass通过下面这些字符串来控制加密，具体流程我就不分析了(主要是通过字符索引来对输入进行与非等操作，这一过程是可逆的)，直接上Z3

```
key=["aaCCzCzzMMaMCMa","bbMMyMyyZZbZMZb","ccCCxCxxJJcJCJc","ddQQwQwwMMdMQMd",
     "eeYYvYvvKKeKYKe","ffHHuHuuCCfCHCf","ggDDtDttKKgKDKg",
    "hhDDsDssOOhODOh","iiCCrCrrIIiICIi","jjOOqOqqDDjDODj","kkSSpSppEEkESEk",
    "llXXoXooTTlTXTl","mmLLnLnnVVmVLVm","nnII_I__HHnHIHn","ooOO_O__CCoCOCo",
    "ppFF_F__NNpNFNp","qqDD_D__BBqBDBq",
    "rrTT_T__BBrBTBr","ssJJ_J__CCsCJCs","ttGG_G__XXtXGXt",
    "uuDD_D__OOuODOu","vvUU_U__JJvJUJv",
    "wwOO_O__SSwSOSw","xxRR_R__SSxSRSx","yyEE_E__KKyKEKy",
    "zzJJ_J__TTzTJTz"]
```

```
from z3 import*

s=Solver()
input=[BitVec(f"input[{i}]",8) for i in range(26)]
enc=[0x64, 0x04, 0x65, 0x0F, 0x2C, 0x5D, 0x39, 0x23, 0x23, 0x00, 0x16, 0x05, 0
    0xA0, 0xB3, 0x93, 0xA9, 0x92, 0xA0, 0xAF, 0xCB, 0x8C, 0xCA]

data=[0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
    0x00,
    0x64, 0x04, 0x65, 0x0F, 0x2C, 0x5D, 0x39, 0x23, 0x23, 0x00, 0x16, 0x05, 0x
    0xA0, 0xB3, 0x93, 0xA9, 0x92, 0xA0, 0xAF, 0xCB, 0x8C, 0xCA]

for i in range(97,123):
    data[i]=input[i-97]

key=["aaCCzCzzMMaMCMa","bbMMyMyyZZbZMZb","ccCCxCxxJJcJCJc","ddQQwQwwMMdMQMd",
     "eeYYvYvvKKeKYKe","ffHHuHuuCCfCHCf","ggDDtDttKKgKDKg",
    "hhDDsDssOOhODOh","iiCCrCrrIIiICIi","jjOOqOqqDDjDODj","kkSSpSppEEkESEk",
    "llXXoXooTTlTXTl","mmLLnLnnVVmVLVm","nnII_I__HHnHIHn","ooOO_O__CCoCOCo",
    "ppFF_F__NNpNFNp","qqDD_D__BBqBDBq",
    "rrTT_T__BBrBTBr","ssJJ_J__CCsCJCs","ttGG_G__XXtXGXt",
    "uuDD_D__OOuODOu","vvUU_U__JJvJUJv",
    "wwOO_O__SSwSOSw","xxRR_R__SSxSRSx","yyEE_E__KKyKEKy",
    "zzJJ_J__TTzTJTz"]

def init_data(key):
```

```
    for i in range(len(key)):
        data[i+1]=key[i]
    #print(data)

def init():
    data[0]=1
    for i in range(5):
        index=ord(data[data[0]+1])
        index1=ord(data[data[0]])
        index2=ord(data[data[0]+2])
        data[0]+=3
        data[index2]=~(data[index]&data[index1])

for i in range(26):
    s.add(data[i+97]>=32,data[i+97]<=127)

for i in range(26):
    init_data(key[i])
    init()

for i in range(26):
    s.add(data[i+97]&0xff==enc[i])

if s.check()==sat:
    m=s.model()
    for i in range(26):
        #as_long将input[i]由Bitvec转为int
        print(chr(m[input[i]].as_long()),end='')
else:
    print("Unsat")

```

getflag：QwQ_s0oOo_simple_LlVm_P4s5

1

Minilinux

考点

魔改AES、Linux内核

Linux下编写和加载 .ko 文件(驱动模块文件)_ko 内核_worthsen的博客-CSDN博客

解题过程

虽然说是内核，但是做题的时候完全没涉及到相关知识(赛后再详细问问出题思路吧)

GetKey

copy_from_user获取用户输入的48位字符

obfs_decode函数对字符串中的每个字符+16

然后和我们输入的前十六位进行比对

```
22  input_1 = input;
23  copy_from_user(input, v2, 48LL);
24  v4 = obfs_decode("\"R'U!%!&\"(QUT\"Q&QRV'!%(( )SV$V#S");
25  *(_QWORD *)s = 0LL;
26  key = (_QWORD *)v4;
27  v6 = (__int64 *)s;
28  v12 = 0LL;
29  v13 = 0LL;
30  v14 = 0LL;
31  do
32  {
33    input_data = (unsigned __int8)*input_1;
34    str = (char *)v6;
35    v6 = (__int64 *)((char *)v6 + 2);
36    ++input_1;
37    sprintf(str, "%02x", input_data);
38  }
39  while ( &v15 != (unsigned __int64 *)v6 );
40  if ( *key != *(_QWORD *)s )
41  {
42    printk(&unk_10AF);
43    return 0xFFFFFFFFFFFFFFEALL;
44  }
45  printk(&unk_1089);
```

魔改AES

将输入后32位传入进行加密，前十六位作为key

```
46  if ( Enc(&input[16], input) )
```

```
9
10  _fentry__(input, key);
11  v8 = __readgsqword(0x28u);
12  *(_QWORD *)&v5 = 0x5A96B0813E935426LL;
13  *((_QWORD *)&v5 + 1) = 0x86E97215021B2394LL;
14  v6 = 0xCA67F9B2C8B5F4C9LL;
15  v7 = 0xB3E603429B5AFA0ALL;
16  memset(v4, 0, sizeof(v4));
17  Init_key(v4, key);
18  sub_190(input, (__int64)v4);
19  sub_190(input + 16, (__int64)v4);
20  v2 = *(_OWORD *)input != v5 || *((_QWORD *)input + 2) != v6 || *((_QWORD *)input + 3) != v7;
21  return !v2;
```

通过和AES加密流程比对会发现**在轮密钥加的过程中异或0x42**

```
      v8 = (int)v6;
      v9 = v2[v6] ^ v3[v6];
      if ( v7 > 3 )
      {
        v120 = v5;
        v129 = v2;
        v132 = v3;
        v140 = v4;
        _ubsan_handle_out_of_bounds(&off_1C60, v7);
        v5 = v120;
        v2 = v129;
        v3 = v132;
        v4 = v140;
        v8 = (int)v6;
      }
      if ( v8 > 3 )
      {
        v130 = v5;
        v133 = v2;
        v141 = v3;
        v151 = v4;
        _ubsan_handle_out_of_bounds(&off_1C20, v8);
        v5 = v130;
        v2 = v133;
        v3 = v141;
        v4 = v151;
      }
      v3[v6++] = v9 ^ 0x42;
    }
```

改一下脚本即可得到flag：miniLctf{C0ngr4ts_l1nux_h@ck3r!}

修改如下

```
static void AddRoundKey(uint8_t round, state_t* state, const uint8_t* RoundKey
{
  uint8_t i,j;
  for (i = 0; i < 4; ++i)
  {
    for (j = 0; j < 4; ++j)
    {
        (*state)[i][j] ^= RoundKey[(round * Nb * 4) + (i * Nb) + j] ^ 0x42;
    }
  }
}
```

1