start_main

在程序运行中，main函数是运行的起点，但是这是对于用户而言的，对于系统而言，在main执行之前需要额外的操作，比如分配资源、内存等等，所以这里就可以插入点东西。main函数是C程序的开始吗? - 知乎 (zhihu.com)→可以简单看下这个了解一下。

之后就是分析。这里涉及的知识点就是base64换表和解密，然后加一个RC4。这里可以按 shift + F7查看一下 ,init_array段和.fini_array段，这两个中有设置的起始和结束函数。

Name	Start	End	R	W	X	D	L	Align
LOAD	0000000000000000	00000000000009A8	R	.	.	.	L	mempage
.init	0000000000001000	000000000000101B	R	.	X	.	L	dword
LOAD	000000000000101B	0000000000001020	R	.	X	.	L	mempage
.plt	0000000000001020	0000000000001110	R	.	X	.	L	para
.plt.got	0000000000001110	0000000000001120	R	.	X	.	L	para
.plt.sec	0000000000001120	0000000000001200	R	.	X	.	L	para
.text	0000000000001200	0000000000001FA3	R	.	X	.	L	para
LOAD	0000000000001FA3	0000000000001FA4	R	.	X	.	L	mempage
.fini	0000000000001FA4	0000000000001FB1	R	.	X	.	L	dword
.rodata	0000000000002000	000000000000203A	R	.	.	.	L	dword
LOAD	000000000000203A	000000000000203C	R	.	.	.	L	mempage
.eh_frame_hdr	000000000000203C	00000000000020C8	R	.	.	.	L	dword
.eh_frame	00000000000020C8	00000000000022D8	R	.	.	.	L	qword
.init_array	0000000000003D40	0000000000003D50	R	W	.	.	L	qword
.fini_array	0000000000003D50	0000000000003D60	R	W	.	.	L	qword
LOAD	0000000000003D60	0000000000003F50	R	W	.	.	L	mempage
.got	0000000000003F50	0000000000004000	R	W	.	.	L	qword
.data	0000000000004000	00000000000041F0	R	W	.	.	L	align_32
LOAD	00000000000041F0	0000000000004200	R	W	.	.	L	mempage
.bss	0000000000004200	0000000000004360	R	W	.	.	L	align_32
extern	0000000000004360	00000000000043F8	?	?	?	.	L	qword

对于 .init_arry 段，这里的sub_1DF1就是比main先执行的函数了，这里就是设置随机数种子，然后将标准base64表循环左移v1形成新的表，然后通过sub_175A进行base64加密，这里加密的数据off_41E8其实就是下面main函数中RC4的密钥，所以这里在main函数执行之前，密钥就已经改变了。当然，动调是可以看出来的。

```
.init_array:0000000000003D40                    _init_array segment qword public 'DATA' use64
.init_array:0000000000003D40                    assume cs:_init_array
.init_array:0000000000003D40                    ;org 3D40h
.init_array:0000000000003D40 E0 12 00 00 00 00 00 00  off_3D40 dq offset sub_12E0          ; DATA XREF: LOAD:0000000000000168↑o
.init_array:0000000000003D40                                                              ; LOAD:00000000000002F0↑o
.init_array:0000000000003D48 F1 1D 00 00 00 00 00 00  dq offset sub_1DF1
.init_array:0000000000003D48                    _init_array ends
.init_array:0000000000003D48
.fini_array:0000000000003D50                    ; ELF Termination Function Table
```

```
1  char *sub_1DF1()
2  {
3    char *result; // rax
4    int v1; // [rsp+Ch] [rbp-4h]
5
6    srand(114514u);
7    v1 = rand() % 64;
8    sub_16CD(s, v1);
9    result = sub_175A(off_41E8);
10   off_41E8 = result;
11   return result;
12 }
```

再到主函数main，这里就是输入flag，然后对于输入的值进行 base64解密(就是sub_1B98进行的操作，注意这里表进行变更了)，最后再来一个 RC4的解密(sub_15CB，它的key也被变了)。

```
__int64 __fastcall main(int a1, char **a2, char **a3)
{
  puts("Please input your flag:");
  fgets(byte_4220, 40, stdin);
  byte_4220[strcspn(byte_4220, "\n")] = 0;
  if ( strlen(byte_4220) != 36 )
  {
    puts("Wrong flag length!");
    exit(0);
  }
  qword_4248 = (__int64)sub_1B98(byte_4220);
  sub_15CB((const char *)qword_4248, (__int64)off_41E8);
  return 0LL;
}
```

之后通过 'fini_array' 段再sub_1E47得到了最后的比较位置。这里两个函数sub_1D4F和sub_1DA0只是为了打乱存储的字符串，防止直接shift+F12查找字符串找出来。它只是个简单的异或，不用处理。这里提取密文dword_4180然后再网上推就可以得到base64格式的flag了。

```
int sub_1E47()
{
  const char *v0; // rax
  const char *v2; // rax
  int i; // [rsp+Ch] [rbp-4h]

  for ( i = 0; i <= 25; ++i )
  {
    if ( *(unsigned __int8 *)(qword_4248 + i) != dword_4180[i] )
    {
      v0 = (const char *)sub_1D4F(&unk_4080);
      return printf("%s", v0);
    }
  }
  v2 = (const char *)sub_1DA0(&unk_40C0);
  return printf("%s", v2);
}
```

套一个解题的脚本，最后base64转换后的输出把 | 改成 } 即可

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

unsigned char sbox[256] = {0};
char* base64_table =
"RSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/ABCDEFGHIJKLMNOPQ";
unsigned char data[] = {102, 47, 150, 124, 156, 100, 13, 207, 181, 197,
236, 146, 119, 151, 171, 135, 227, 189, 178, 179, 180, 53, 227, 105, 54,
121};
char *key = "lFaUmVp=";
void swap(unsigned char *a, unsigned char *b)
{
    unsigned char tmp = *a;
    *a = *b;
```

```
    *b = tmp;
}
void init_sbox(unsigned char key[])
{
    for (unsigned int i = 0; i < 256; i++) // 赋值
        sbox[i] = i;
    unsigned int keyLen = strlen((char *)key);
    unsigned char Ttable[256] = {0};
    for (int i = 0; i < 256; i++)
        Ttable[i] = key[i % keyLen]; // 根据初始化t表
    for (int j = 0, i = 0; i < 256; i++)
    {
        j = (j + sbox[i] + Ttable[i]) % 256; // 打乱s盒
        swap(&sbox[i], &sbox[j]);
    }
}
void RC4(unsigned char data[], char key[])
{
    unsigned char k, i = 0, j = 0, t;
    init_sbox(key);
    unsigned int dataLen = strlen(data);
    for (unsigned int h = 0; h < dataLen; h++)
    {
        i = (i + 1) % 256;
        j = (j + sbox[i]) % 256;
        swap(&sbox[i], &sbox[j]);
        t = (sbox[i] + sbox[j]) % 256;
        k = sbox[t];
        data[h] ^= k;
    }
}

char *base64_encode(unsigned char *str)
{
    int len = strlen(str);
    char *ans = (char *)malloc((len * 4 / 3 + 4) * sizeof(char));
    int j = 0;
    for (int i = 0; i < len / 3 * 3; i += 3)
    {
        ans[j++] = base64_table[str[i] >> 2];
        ans[j++] = base64_table[(str[i] & 0x3) << 4 | (str[i + 1]) >> 4];
        ans[j++] = base64_table[(str[i + 1] & 0xf) << 2 | (str[i + 2]) >>
6];
        ans[j++] = base64_table[(str[i + 2]) & 0x3f];
    }
    if (len % 3 == 1)
    {
        int pos = len / 3 * 3;
        ans[j++] = base64_table[str[pos] >> 2];
        ans[j++] = base64_table[(str[pos] & 0x3) << 4];
        ans[j++] = '=';
        ans[j++] = '=';
    }
    else if (len % 3 == 2)
```

```
    {
        int pos = len / 3 * 3;
        ans[j++] = base64_table[str[pos] >> 2];
        ans[j++] = base64_table[(str[pos] & 0x3) << 4 | (str[pos + 1]) >>
4];
        ans[j++] = base64_table[(str[pos + 1] & 0xf) << 2];
        ans[j++] = '=';
    }
    ans[j] = '\0';
    return ans;
}
int find_index(char c)
{
    char *ptr = strchr(base64_table, c);
    if (ptr)
    {
        return ptr - base64_table;
    }
    else
    {
        return -1;
    }
}

unsigned char *base64_decode(char *str)
{
    int len = strlen(str);
    unsigned char *ans = (unsigned char *)malloc((len * 3 / 4 + 1) *
sizeof(char));
    int j = 0;
    for (int i = 0; i < len; i += 4)
    {
        ans[j++] = (find_index(str[i]) << 2) | (find_index(str[i + 1]) >>
4);
        if (str[i + 2] != '=')
        {
            ans[j++] = ((find_index(str[i + 1]) & 0xf) << 4) |
(find_index(str[i + 2]) >> 2);
        }
        if (str[i + 3] != '=')
        {
            ans[j++] = ((find_index(str[i + 2]) & 0x3) << 6) |
find_index(str[i + 3]);
        }
    }
    return ans;
}

int main(){
    RC4(data, key);
    char *s = base64_encode(data);
    // for(int i = 0; i < 26; i++){
    //     printf("%0x ", data[i]);
    // }
```

```
    printf("%s\n", s);
    base64_table =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";
    unsigned char *ans = base64_decode(s);
    printf("%s\n", ans);
}
```

ez_ptrace

这个题如它的题目，就是考察ptrace的知识点。这里首先查看main函数。它就是先检查程序运行的参数，然后没有的话就不能执行，然后对参数添加一个'./'形成一个新的字符串，可以看出这里应该是一个路径。

```
int __cdecl sub_14BA(int a1, int a2)
{
  char s[256]; // [esp+0h] [ebp-10Ch] BYREF
  __pid_t v4; // [esp+100h] [ebp-Ch]
  int *v5; // [esp+104h] [ebp-8h]

  v5 = &a1;
  if ( a1 != 2 )
  {
    fwrite("ERROR: Please enter correct parameters\n", 1u, 0x27u, stderr);
    exit(1);
  }
  sprintf(s, "./%s", *(_DWORD *)(a2 + 4));
  sub_124D();
  v4 = fork();
  if ( v4 )
  {
    if ( v4 <= 0 )
    {
      perror("fork");
      return -1;
    }
    sub_145F(v4);
  }
  else
  {
    sub_1424(s);
  }
  return 0;
}
```

然后查看下面的函数sub_124D，这里就是简单的打开文件'./son'，然后把一些内容写进去，最后还用了chmod赋予权限，根据权限0x1FF，在八进制中就是777，可以猜测这是个可执行文件，然后继续往下看。

```
void sub_124D()
{
  FILE *stream; // [esp+0h] [ebp-28h]
  int jj; // [esp+4h] [ebp-24h]
  int ii; // [esp+8h] [ebp-20h]
  int n; // [esp+Ch] [ebp-1Ch]
  int m; // [esp+10h] [ebp-18h]
  int k; // [esp+14h] [ebp-14h]
  int j; // [esp+18h] [ebp-10h]
  int i; // [esp+1Ch] [ebp-Ch]

  stream = fopen("./son", "wb+");
  if ( !stream )
    perror("Error opening file");
  for ( i = 0; i <= 1207; ++i )
    fputc(byte_4020[i] ^ 0x67, stream);
  for ( j = 0; j <= 2887; ++j )
    fputc(0, stream);
  for ( k = 0; k <= 1487; ++k )
    fputc(byte_44E0[k] ^ 0x30, stream);
  for ( m = 1; m <= 2608; ++m )
    fputc(0, stream);
  for ( n = 0; n <= 419; ++n )
    fputc(byte_4AC0[n], stream);
  for ( ii = 1; ii <= 3352; ++ii )
    fputc(0, stream);
  for ( jj = 0; jj <= 1763; ++jj )
    fputc(byte_4C80[jj], stream);
  fclose(stream);
  if ( chmod("./son", 0x1FFu) )
    perror("Error changing file permissions");
}
```

之后使用fork()创建了父子进程，父进程执行sub_145F，子进程执行sub_1424，查看这两个函数。再根据ptrace的知识，这里子进程使用PTRACE_TRACEME表示自己可以被父进程跟踪，然后执行了这个path文件，结合上文，这个执行的就是我们输入的参数，在结合那个可执行文件的名称，大致猜到我们就是要输入son，来让子进程执行这个文件。再次看父进程，它在最终子进程的过程中使用了PTRACE_POKEDATA来改变子进程的一个数值dword_4C64为49，之后再让子进程继续执行。

目前根据已有知识，可以知道该程序就是输入正确参数起一个子进程，对子进程的内存数据进行更改后把控制权返还给子进程，子进程负责加密操作。

```
__pid_t __cdecl sub_145F(int a1)
{
  char stat_loc[8]; // [esp+Ch] [ebp-Ch] BYREF

  wait(stat_loc);
  ptrace(PTRACE_POKEDATA, a1, dword_4C64, 49);
  ptrace(PTRACE_CONT, a1, 0, 0);
  return wait(0);
}
```

```
int __cdecl sub_1424(char *path)
{
  ptrace(PTRACE_TRACEME, 0, 0, 0);
  return execl(path, path, 0);
}
```

然后让程序执行时在子进程执行程序前下断点，查看本地的son文件信息。查看main函数，这里先删除本地的son文件，让文件信息只在内存中存在，再进行普通的XXTEA加密。

查看密钥的位置，发现0x6000404C位置的密钥和之前父进程执行PTRACE_POKEDATA修改的dword_4C64表示地址一样，判断父进程修改了这里的数值，所以进行XXTEA解密时需要更改最后一个数值为49来解密。然后就是正常的XXTEA解密。

```
.data:6000403F 04                       db    4
.data:60004040 74                       unk_60004040 db  74h ; t                ;
.data:60004041 00                       db    0
.data:60004042 00                       db    0
.data:60004043 00                       db    0
.data:60004044 71                       db  71h ; q
.data:60004045 00                       db    0
.data:60004046 00                       db    0
.data:60004047 00                       db    0
.data:60004048 6C                       db  6Ch ; l
.data:60004049 00                       db    0
.data:6000404A 00                       db    0
.data:6000404B 00                       db    0
.data:6000404C 21                       db  21h ; !
.data:6000404D 00                       db    0
.data:6000404E 00                       db    0
.data:6000404F 00                       db    0
.data:6000404F                          _data ends
.data:6000404F
.bss:60004050                           ; ========================================
```

贴一个脚本

```
#include <stdio.h>
#define ut32 unsigned int
#define delta 0x9e3779b9
#define MX (((z >> 5 ^ y << 2) + (y >> 3 ^ z << 4)) ^ ((sum ^ y) + (key[(p
& 3) ^ e] ^ z)))
```

```
void XXTea_Decrypt(ut32 *enc, ut32 n, ut32 *key);
void output(ut32 *m, ut32 len);

void output(ut32 *m, ut32 len)
{
    for (int i = 0; i < len; i++)
        printf("0x%08x, ", m[i]);
    printf("\n");
}

void XXTea_Decrypt(ut32 *enc, ut32 n, ut32 *key)
{
    ut32 y, z, sum;
    ut32 e, rounds;
    int p;
    rounds = 6 + 52 / n;
    sum = delta * rounds;

    do
    {
        e = (sum >> 2) & 3;

        for (p = n - 1; p > 0; p--)
        {
            y = enc[(p + 1) % n];
            z = enc[(p - 1) % n];
            enc[p] -= (((z >> 5 ^ y << 2) + (y >> 3 ^ z << 4)) ^ ((sum ^ y)
+ (key[(p & 3) ^ e] ^ z)));
        }
        y = enc[1];
        z = enc[n - 1];
        enc[p] -= (((z >> 5 ^ y << 2) + (y >> 3 ^ z << 4)) ^ ((sum ^ y) +
(key[(p & 3) ^ e] ^ z)));
        sum -= delta;
    } while (--rounds);
}
void hex_to_string_little_endian(const ut32 *input, size_t len, char
*output)
{
  for (size_t i = 0; i < len; ++i)
  {
    for (size_t j = 0; j < 4; ++j)
    {
      output[i * 4 + j] = (char)(input[i] >> (j * 8));
    }
  }
  output[len * 4] = '\0';
}

int main()
{
    ut32 enc[8] = {0x0913bc10, 0x7ad2de6f, 0xec01321e, 0x33c2a85d,
0x4476c2de, 0x59714e3b, 0xf5b45769, 0x04b4e6ec};
    ut32 k[4] = {116, 113, 108, 49};
```

```
    XXTea_Decrypt(enc, 8, k);
    output(enc, 8);
    char str[33];
    hex_to_string_little_endian(enc, 8, str);
    printf("%s\n", str);
    return 0;
}
```