

# 攻防世界逆向入门题之no-strings-attached

原创

沐一·林 于 2021-08-13 11:35:30 发布 161 收藏

分类专栏: [CTF 逆向](#) 文章标签: [unctf](#)

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY-SA](#) 版权协议, 转载请附上原文出处链接和本声明。

本文链接: [https://blog.csdn.net/xiao\\_\\_1bai/article/details/119612880](https://blog.csdn.net/xiao__1bai/article/details/119612880)

版权



CTF 同时被 2 个专栏收录

167 篇文章 6 订阅

订阅专栏



逆向

95 篇文章 6 订阅

订阅专栏

## 攻防世界逆向入门题之no-strings-attached

继续开启全栈梦想之逆向之旅~

这题是攻防世界逆向入门题的no-strings-attached

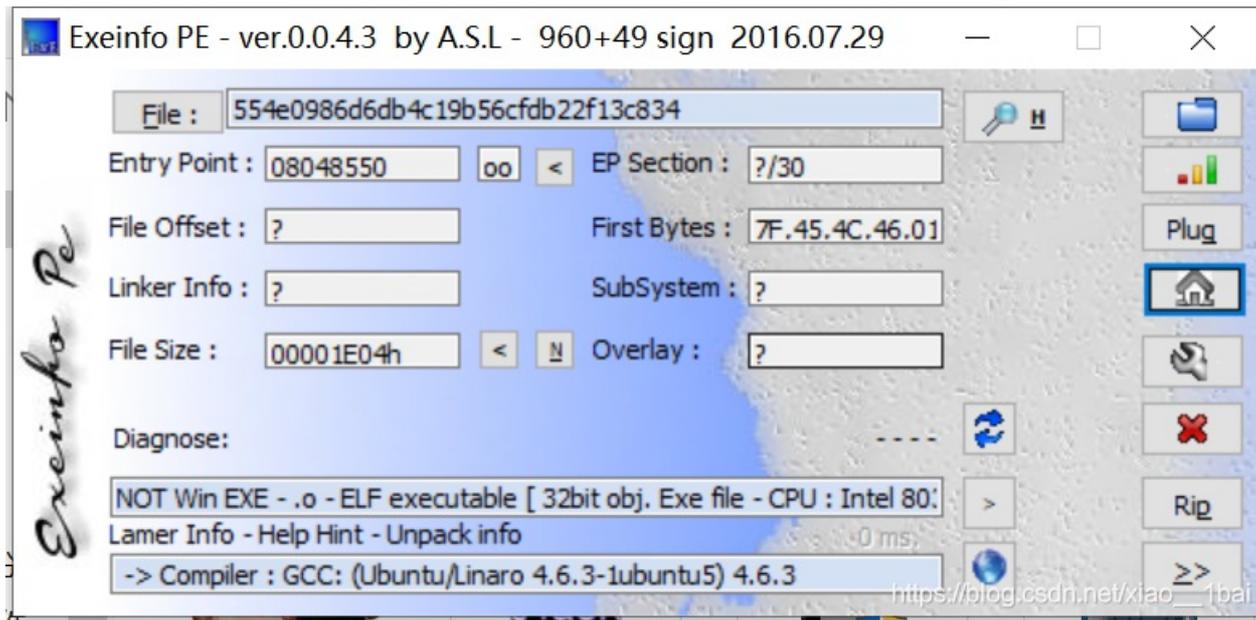
The screenshot shows a CTF challenge interface. At the top, there is a navigation bar with a '返回' (Return) button and a timer showing '本题用时: 46分2秒'. The challenge title is 'no-strings-attached' with a thumbs-up icon and '47' likes. Below the title, it says '最佳Writeup由Syclover • lingze提供'. There are two buttons: 'WP' and '建议'. The challenge details include: '难度系数: 4.0' (Difficulty: 4.0), '题目来源: 9447 CTF 2014' (Source: 9447 CTF 2014), '题目描述: 菜鸡听说有的程序运行就能拿Flag?' (Description: A noob heard that some programs can get a flag by running?), '题目场景: 暂无' (Scenario: None), and '题目附件: 附件1' (Attachments: Attachment 1). The URL 'https://blog.csdn.net/xiao\_\_1bai' is visible in the bottom right corner.

下载附件:

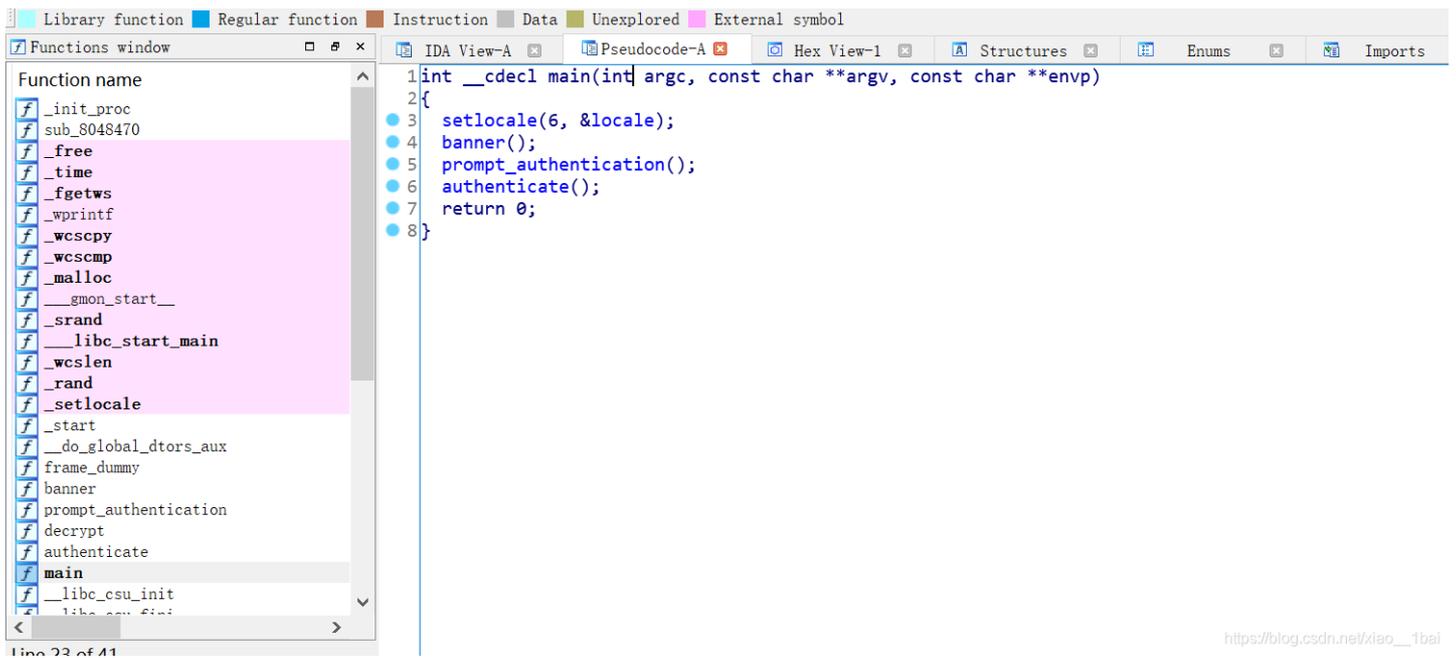


554e0986d |  
6db4c19b5  
6cfdb22f13  
c834

扔入Exeinfo中查壳和其他信息:



如图所示, 32位ELF的linux文件, 照例扔如IDA32位中查看代码信息, 跟进Main函数:



看到四个函数, 由于才疏学浅, 以为flag不在此, 还去查看了一下strings窗口:

The screenshot shows the strings window in IDA Pro, displaying a list of loaded strings. The table has columns for Address, Length, Type, and String.

Address	Length	Type	String
LOAD:080...	00000013	C	/lib/ld-linux. so. 2
LOAD:080...	0000000F	C	__gmon_start__
LOAD:080...	0000000A	C	libc. so. 6
LOAD:080...	0000000F	C	_IO_stdin_used
LOAD:080...	0000000A	C	setlocale
LOAD:080...	00000006	C	srand
LOAD:080...	00000005	C	time
LOAD:080...	00000006	C	stdin
LOAD:080...	00000007	C	fgetws
LOAD:080...	00000007	C	wscmp
LOAD:080...	00000007	C	...

LOAD:080...	00000007	C	malloc
LOAD:080...	00000007	C	wscpy
LOAD:080...	00000008	C	wprintf
LOAD:080...	00000012	C	__libc_start_main
LOAD:080...	00000007	C	wcslen
LOAD:080...	00000005	C	free
LOAD:080...	0000000A	C	GLIBC_2.2
LOAD:080...	0000000A	C	GLIBC_2.0
.eh_fram...	00000005	C	;*2\$\

[https://blog.csdn.net/xiao\\_\\_1bai](https://blog.csdn.net/xiao__1bai)

也没有flag字眼，有点懵(还是没觉得main的四个函数有问题，还是太菜了啊)。查了查资料，说flag操作就在这四个函数里，于是有回头去看这四个函数，先看导入表，看那些是自带的函数：

Address	Ordinal	Name
0804A04C		free
0804A050		time
0804A054		fgetws
0804A058		wprintf
0804A05C		wscpy
0804A060		wscmp
0804A064		malloc
0804A068		srand
0804A06C		__libc_start_main
0804A070		wcslen
0804A074		rand
0804A078		setlocale
0804A07C		__gmon_start__

[https://blog.csdn.net/xiao\\_\\_1bai](https://blog.csdn.net/xiao__1bai)

可以看见第一个setlocale是自带的函数，第二第三个双击跟踪进去是打印函数，banner（横幅），猜测应该是打印开头信息的，那么就剩下第四个函数了，双击查看内容：

```

1 void authenticate()
2 {
3     int ws[8192]; // [esp+1Ch] [ebp-800Ch]
4     wchar_t *s2; // [esp+801Ch] [ebp-Ch]
5
6     s2 = decrypt(&s, &dword_8048A90);
7     if ( fgetws(ws, 0x2000, stdin) )
8     {
9         ws[wcslen(ws) - 1] = 0;
10        if ( !wcscmp(ws, s2) )
11            wprintf((int)&unk_8048B44);
12        else
13            wprintf((int)&unk_8048BA4);
14    }
15    free(s2);
16}

```

[https://blog.csdn.net/xiao\\_\\_1bai](https://blog.csdn.net/xiao__1bai)

这里有个decrypt函数，中文名是加密，不在导入表中说明不是系统函数，后面的if判断条件是输入，还有个比较的wcscmp函数，后面两个wprintf分别是success 和access这些成功和拒绝的字符串地址。

fgetws函数是从输入流stdin中获取0x2000个字符给ws，也就是说s2是关键了，s2由decrypt函数得出，decrypt是用户自定义函数，在这里学到了非系统函数的英文名会是题目给的暗示，所以这里是加密操作后与输入的比较，只要输入后与加密后的s2一样就会打印success或access这些字符串，那flag自然也在加密函数中了。

双击跟进：

```
1 wchar_t *__cdecl decrypt(wchar_t *s, wchar_t *a2)
2 {
3     size_t v2; // eax
4     signed int v4; // [esp+1Ch] [ebp-1Ch]
5     signed int i; // [esp+20h] [ebp-18h]
5     signed int v6; // [esp+24h] [ebp-14h]
7     signed int v7; // [esp+28h] [ebp-10h]
3     wchar_t *dest; // [esp+2Ch] [ebp-Ch]
3
3     v6 = wcslen(s);
1     v7 = wcslen(a2);
2     v2 = wcslen(s);
3     dest = (wchar_t *)malloc(v2 + 1);
4     wcscpy(dest, s);
5     while ( v4 < v6 )
5     {
7         for ( i = 0; i < v7 && v4 < v6; ++i )
3             dest[v4++] -= a2[i];
3     }
3     return dest;
1 }
```

[https://blog.csdn.net/xiao\\_\\_1bai](https://blog.csdn.net/xiao__1bai)

没学过wchar.h库，没办法，继续查资料(wp)，攻防世界admin官方的wp是用GDB的动态调试，我以前也没用过gdb，也才发现linux也可以动态调试，真的由于太菜长见识了：

这里我们有两种思维方式：

第一种就是跟进decrypt然后分析它的运算逻辑，然后，自己写脚本，得到最后的flag

第二种就涉及逆向的另一种调试方式，及动态调试，这里我就用动态调试了，之前的一直是静态调试

0x03.GDB动态调试

```
root@kali:~/mnt/hgfs# gdb ./no_strings_attached
GNU gdb (Debian 8.2.1-2) 8.2.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
```

[https://blog.csdn.net/xiao\\_\\_1bai](https://blog.csdn.net/xiao__1bai)

学GDB学了两天还是三天，然后才跟着admin的wp复现了一遍，这里说一下自己的分析：

由于这种题是和用户输入的比较的，也就是说flag就在s2里面，我们可以在内存调试中提取s2的值，然后解密即可得到flag。(通常s2就是flag，因为如果s2还是加密的flag的话就不用完了)

我还尝试print s2指令输出变量s2的值，因为我以为和IDA显示的一样，flag赋值给了s2,后来才想起IDA是根据自己的规则给无法解析变量名赋值的，也就是说在IDA里变量是s2这个名字，但是实际上程序里并没有s2这个变量名,所以只能查看寄存器了，毕竟函数是先返回到eax寄存器中再移动到变量中的。

还有就是admin的wp中给的是n指令然后查看eax寄存器的值，可是n指令执行的是一行高级语言命令，而ni和si才是单步执行一条汇编指令，所以不要调着调着跳过对应指令都不知道。

还有就是这里虽然是decrypt产出flag后赋值给了s2，但是双击s2跟踪显示的是s2初始的地址和值，而s2初始并没有什么东西，decrypt函数是用初始有值的&s进行加密操作后才产出flag赋值给s2的，所以不能用双击跟踪s2初始值的方式得到flag。

&s双击后跟进的字符串值：

```
5 |
6 | s2 = decrypt(&s, &dword_8048A90);
7 | if ( fgetws(ws, 0x2000, stdin) )
8 | {
9 |     ws[wcslen(ws) - 1] = 0;
```

```
.rodata:08048AA8 s dd 143Ah
.rodata:08048AAC db 36h ; 6
.rodata:08048AAD db 14h
.rodata:08048AAE db 0
.rodata:08048AAF db 0
.rodata:08048AB0 db 37h ; 7
.rodata:08048AB1 db 14h
.rodata:08048AB2 db 0
.rodata:08048AB3 db 0
.rodata:08048AB4 db 3Bh ; ;
.rodata:08048AB5 db 14h
.rodata:08048AB6 db 0
.rodata:08048AB7 db 0
.rodata:08048AB8 db 80h
.rodata:08048AB9 db 14h
.rodata:08048ABA db 0
.rodata:08048ABB db 0
.rodata:08048ABC db 7Ah ; z
.rodata:08048ABD db 14h
.rodata:08048ABE db 0
.rodata:08048ABF db 0
.rodata:08048AC0 db 71h ; q
.rodata:08048AC1 db 14h
.rodata:08048AC2 db 0
.rodata:08048AC3 db 0
.rodata:08048AC4 db 78h ; x
.rodata:08048AC5 db 14h
```

这里是引用  
GDB动态调试

```
root@kali:~/mnt/hgfs# gdb ./no_strings_attached
GNU gdb (Debian 8.2.1-2) 8.2.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
```

```
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./no_strings_attached...(no debugging symbols found)...done
https://blog.csdn.net/xiao__1bai
```

gdb ./no\_strings\_attached 将文件加载到GDB中

```
(gdb) b decrypt
Breakpoint 1 at 0x804865c
(gdb)
```

之前通过IDA，我们知道关键函数是decrypt,所以我们将断点设置在decrypt处，b在GDB中就是下断点的意思，及在decrypt处下断点

```
Breakpoint 1 at 0x804865c
(gdb) r
Starting program: /mnt/hgfs/no_strings_attached
Welcome to cyber malware control software.
Currently tracking 1878339819 bots worldwide

Breakpoint 1, 0x804865c in decrypt ()
```

我们要的是经过decrypt函数，生成的字符串，所以我们这里就需要运行一步，GDB中用n来表示运行一步高级语言代码

```
(gdb) n
Single stepping until exit from function decrypt
which has no line number information.
0x8048725 in authenticate ()
```

然后我们就需要去查看内存了，去查找最后生成的字符串

```
.text:08048708 ; __unwind {
.text:08048708      push    ebp
.text:08048709      mov     ebp, esp
.text:0804870B      sub     esp, 8028h
.text:08048711      mov     dword ptr [esp+4], offset dword_8048A90 ; wchar_t *
.text:08048719      mov     dword ptr [esp], offset s ; s
.text:08048720      call   decrypt
.text:08048725      mov     [ebp+s2], eax
.text:08048728      mov     eax, ds:stdin@@GLIBC_2_0
.text:0804872D      mov     [esp+8], eax ; stream
.text:08048731      mov     dword ptr [esp+4], 2000h ; n
.text:08048739      lea    eax, [ebp+ws]
.text:0804873F      mov     [esp], eax ; ws
.text:08048742      call   feetws
https://blog.csdn.net/xiao__1bai
```

通过IDA生成的汇编指令，我们可以看出进过decrypt函数后，生成的字符串保存在EAX寄存器中，所以，我们在GDB就去查看eax寄存器的值

```
root@kali: /mnt/hgfs
File Edit View Search Terminal Help
--Type <RET> for more, q to quit, c to continue without pa
Quit
(gdb) x/200wx $eax
```

```

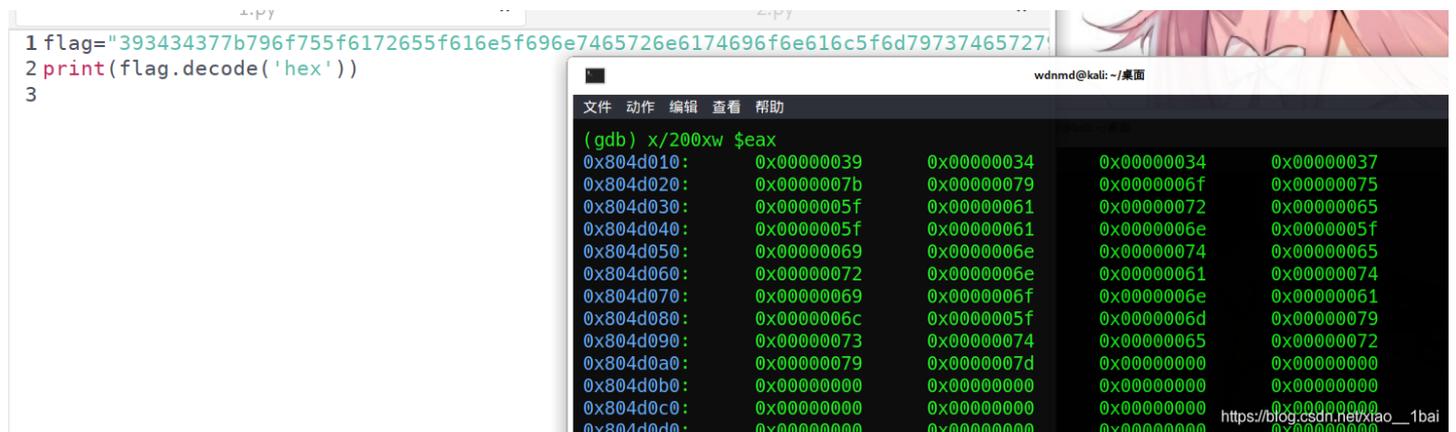
0x804e800: 0x00000039 0x00000034 0x00000034 0x00000034
0x804e810: 0x0000007b 0x00000079 0x0000006f 0x0000006f
0x804e820: 0x0000005f 0x00000061 0x00000072 0x00000072
eaxpad 0x804e830: 0x0000005f 0x00000061 0x0000006e 0x0000006e
0x804e840: 0x00000069 0x0000006e 0x00000074 0x00000074
0x804e850: 0x00000072 0x0000006e 0x00000061 0x00000061
0x804e860: 0x00000069 0x0000006f 0x0000006e 0x0000006e
0x804e870: 0x0000006c 0x0000005f 0x0000006d 0x0000006d
0x804e880: 0x00000073 0x00000074 0x00000065 0x00000065
0x804e890: 0x00000079 0x0000007d 0x00000000 0x00000000

```

x就是用来查看内存中数值的，后面的200代表查看多少个，wx代表是以word字节查看，\$eax代表的eax寄存器中，在这里我们看到0x00000000，这就证明这个字符串结束了，因为在C中，代表字符串结束的就是“\0”，那么前面的就是经过decrypt函数生成的flag

这里要特别注意一下：操作是面对反汇编低级语言来操作的，所以是对照着内存来操作的！

复现后的结果如图：



这里是内存数，所以不用像小端一样反过来(可能只有我才会傻到反过来吧~)，十六进制数解密后就是flag了：

```

flag="393434377b796f755f6172655f616e5f696e7465726e6174696f6e616c5f6d7973746572797d";
print(flag.decode('hex'))

```

注意，这里请用python2执行，具体原因自己查吧。

补充：之前用的是GDB查看内存，这是一种新方式，那么老方法——用IDA还是要掌握的，这里借鉴了一篇博客的操作来复现，并附上自己的见解：

[https://blog.csdn.net/liuxiaohuai\\_/article/details/110002845?utm\\_medium=distribute.pc\\_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromMachineLearnPai2%7Edefault-3.control&depth\\_1-utm\\_source=distribute.pc\\_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromMachineLearnPai2%7Edefault-3.control](https://blog.csdn.net/liuxiaohuai_/article/details/110002845?utm_medium=distribute.pc_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromMachineLearnPai2%7Edefault-3.control&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromMachineLearnPai2%7Edefault-3.control)

首先回顾前面的话：

由于这种题是和用户输入的比较的，也就是说flag就在s2里面，我们可以在内存调试中提取s2的值，然后解密即可得到flag。

flag在s2内，不用gdb查看内存的话s2就无法得知，但是s2是由decrypt这个加密函数得出的，而这里decrypt传入的加密参数&s和&dword\_8048A90都可以双击跟踪内存查看初始值，而且decrypt的内部构造也有，那么我们直接提取出&s和&dword\_8048A90这两个参数的值，然后仿照decrypt写个一样加密流程的脚本得出的不也是flag吗？

```
s2 = decrypt(&s, &dword_8048A90);
```

```
1 wchar_t *__cdecl decrypt(wchar_t *s, wchar_t *a2)
2 {
3     size_t v2; // eax
4     signed int v4; // [esp+1Ch] [ebp-1Ch]
5     signed int i; // [esp+20h] [ebp-18h]
5     signed int v6; // [esp+24h] [ebp-14h]
7     signed int v7; // [esp+28h] [ebp-10h]
3     wchar_t *dest; // [esp+2Ch] [ebp-Ch]
9
9     v6 = wcslen(s);
L    v7 = wcslen(a2);
2    v2 = wcslen(s);
3    dest = (wchar_t *)malloc(v2 + 1);
4    wcscpy(dest, s);
5    while ( v4 < v6 )
5    {
7        for ( i = 0; i < v7 && v4 < v6; ++i )
3            dest[v4++] -= a2[i];
9    }
9    return dest;
L}
```

[https://blog.csdn.net/xiao\\_\\_1bai](https://blog.csdn.net/xiao__1bai)

事实上的确如此！所以我们去提取&s和&dword\_8048A90的内容，那么这里就涉及提取脚本了：直接给前面博客的脚本：

提取&s

```
addr=0x08048AA8 #数组的地址
arr = []
for i in range(39): #数组的个数
    arr.append(Dword(addr+4* i))
print(arr)
```

```
8048645: using guessed type int prompt_authentication(void);
8048708: using guessed type int authenticate(void);
[5178L, 5174L, 5175L, 5179L, 5248L, 5242L, 5233L, 5240L, 5219L, 5222L, 5235L, 5223L, 5218L, 5221L, 5235L, 5216L, 5227L, 5233L, 5240L, 5226L, 5235L, 5232L, 5220L, 5240L, 5230L, 5232L, 5232L, 5220L, 5232L, 5220L, 5230L, 5243L, 5238L, 5240L, 5226L, 5235L, 5243L, 5248L, 0L]
8048480: using guessed type int __cdecl wprintf( _Dword_);
```

提取&dword\_8048A90:

```
addr=0x08048A90 #数组的地址
arr = []
for i in range(6): #数组的个数
    arr.append(Dword(addr+4* i))
print(arr)
```

```
←  
[5121L, 5122L, 5123L, 5124L, 5125L, 0L]  
80484B0: using guessed type int __cdecl wprintf(_DWORD);
```

这里就是简单地提取数据而已啊，后来我发现其实手工也可以，只要对着地址一个个转为dd再转成十进制即可，都是要用邮件，所以IDA功能还是要看IDA权威指南多学学啊！

```
.rodata:08048A90 dword_8048A90 dd 1401h  
.rodata:08048A94 dd 1402h  
.rodata:08048A98 db 3  
.rodata:08048A99 db 14h  
.rodata:08048A9A db 0  
.rodata:08048A9B db 0  
.rodata:08048A9C db 4  
.rodata:08048A9D db 14h  
.rodata:08048A9E db 0  
.rodata:08048A9F db 0  
.rodata:08048AA0 db 5  
.rodata:08048AA1 db 14h  
.rodata:08048AA2 db 0  
.rodata:08048AA3 db 0  
.rodata:08048AA4 db 0  
.rodata:08048AA5 db 0
```

[https://blog.csdn.net/xiao\\_\\_1bai](https://blog.csdn.net/xiao__1bai)

```

.rodata:08048AA8 s          dd 5178
.rodata:08048AAC          dd 1436h
.rodata:08048AB0          dd 1437h
.rodata:08048AB4          dd 143Bh
.rodata:08048AB8          dd 1480h
.rodata:08048ABC          dd 147Ah
.rodata:08048AC0          dd 1471h
.rodata:08048AC4          dd 1478h
.rodata:08048AC8          dd 1463h
.rodata:08048ACC          dd 1466h
.rodata:08048AD0          dd 1473h
.rodata:08048AD4          dd 1467h
.rodata:08048AD8          dd 1462h
.rodata:08048ADC          dd 1465h
.rodata:08048AE0          dd 1473h
.rodata:08048AE4          dd 1460h
.rodata:08048AE8          dd 146Bh
.rodata:08048AEC          dd 1471h
.rodata:08048AF0          dd 1478h
.rodata:08048AF4          dd 146Ah
.rodata:08048AF8          db 73h ; 1sai

```

[https://blog.csdn.net/xiao\\_\\_1bai](https://blog.csdn.net/xiao__1bai)

然后就是用python仿照decrypt加密流程写脚本了:

原来的:

```

wchar_t *__cdecl decrypt(wchar_t *s, wchar_t *a2)
{
    size_t v2; // eax
    signed int v4; // [esp+1Ch] [ebp-1Ch]
    signed int i; // [esp+20h] [ebp-18h]
    signed int v6; // [esp+24h] [ebp-14h]
    signed int v7; // [esp+28h] [ebp-10h]
    wchar_t *dest; // [esp+2Ch] [ebp-Ch]

    v6 = wcslen(s);
    v7 = wcslen(a2);
    v2 = wcslen(s);
    dest = (wchar_t *)malloc(v2 + 1);
    wcscpy(dest, s);
    while ( v4 < v6 )
    {
        for ( i = 0; i < v7 && v4 < v6; ++i )
            dest[v4++] -= a2[i];
    }
    return dest;
}

```

[https://blog.csdn.net/xiao\\_\\_1bai](https://blog.csdn.net/xiao__1bai)

仿照的(也是前面博客的):

```

s = [5178, 5174, 5175, 5179, 5248, 5242, 5233, 5240, 5219, 5222, 5235, 5223, 5218, 5221, 5235, 5216, 5227, 5233,
5240, 5226, 5235, 5232, 5220, 5240, 5230, 5232, 5232, 5220, 5232, 5220, 5230, 5243, 5238, 5240, 5226, 5235, 524
3, 5248]
a = [5121, 5122, 5123, 5124, 5125]
v6 = len(s)
v7 = len(a )
v2 = len(s)

v4=0
while v4<v6:

    for i in range(0,5):
        if(i<v7 and v4<v6):
            s[v4]-=a[i]
            v4 += 1
        else:
            break
for i in range (38):
    print(chr(s[i]),end="")

```

注意：前面c++中v4++是先赋值后再加，所以到了python中v4+=1就放在赋值后面了。



```

└─$ python 1.py
9447{you_are_an_international_mystery}

```

解毕！敬礼！