逆向入门系列（2）静态分析工具ida的使用

原创

NoOneGroup 于 2018-08-28 21:04:08 发布 7158 收藏 14
分类专栏： 逆向 reverse Linux


逆向 同时被 3 个专栏收录
18 篇文章 3 订阅
订阅专栏


reverse
15 篇文章 0 订阅
订阅专栏

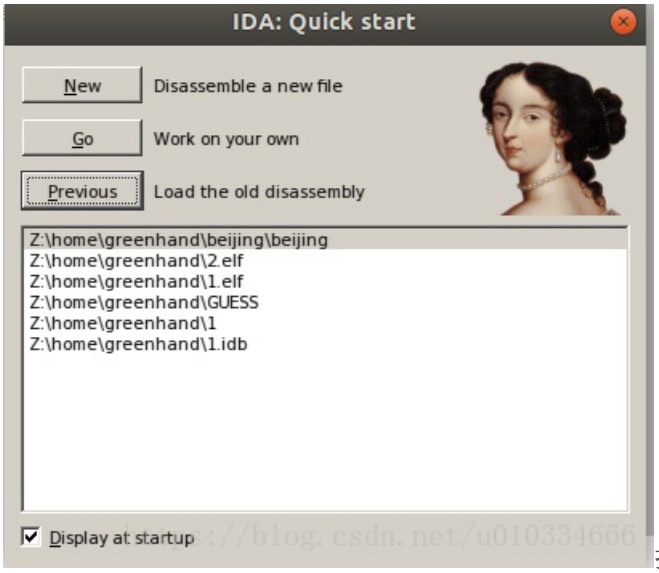

Linux
15 篇文章 0 订阅
订阅专栏

博客已经转移

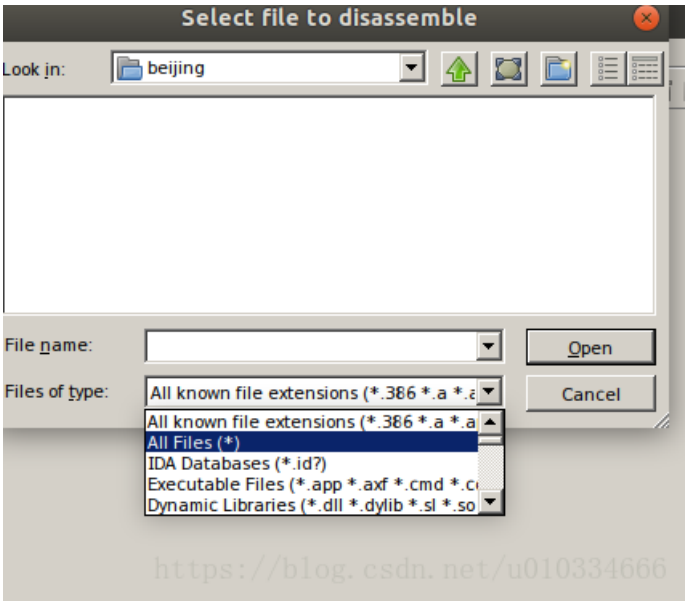
https://noone-hub.github.io/

首

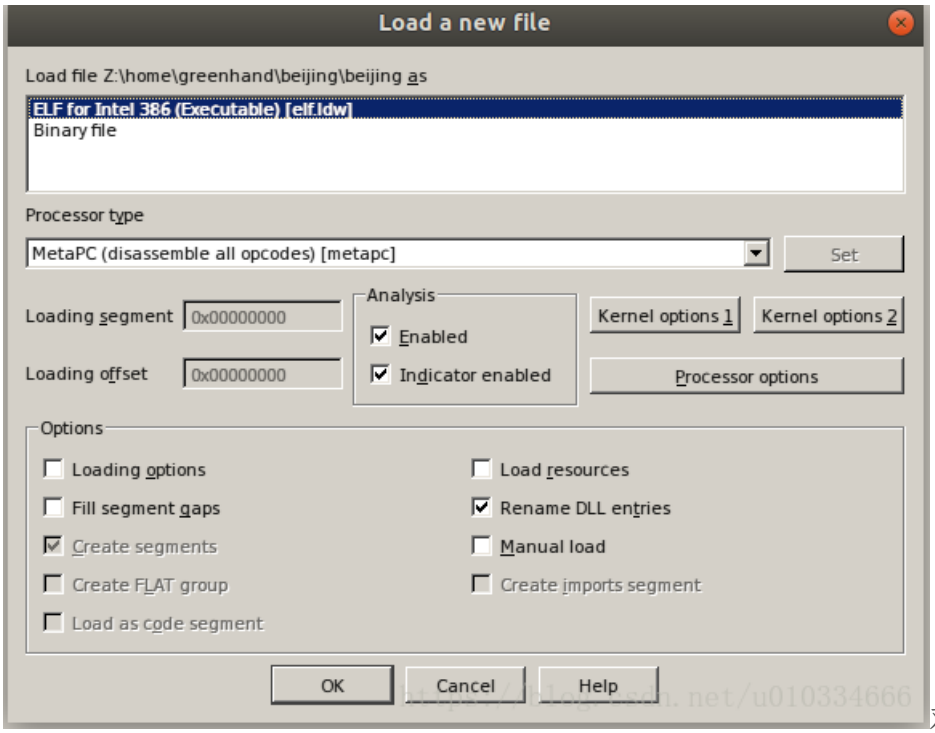
先，ida的下载这个自己可以吧，不会的可以找我要ida6.8版本跟7.0版本，我只有这两个版本，6.8兼容32和64位系统，7.0只兼容64位系统，本文用的是6.8，在linux下，我用wine运行的



打开ida后通常是这个界面，点击new



找到你要的文件，linux下的elf文件要选择all files才能找到，如何判断是elf文件，这我也不谈，可以自己百度，



双击后便是这个界面，他也会给你判断是什么文件，直接点ok就好，进来后不要直接f5，会提示错误的，这里还不是main函数，你f5也没用，f5是转c代码的，

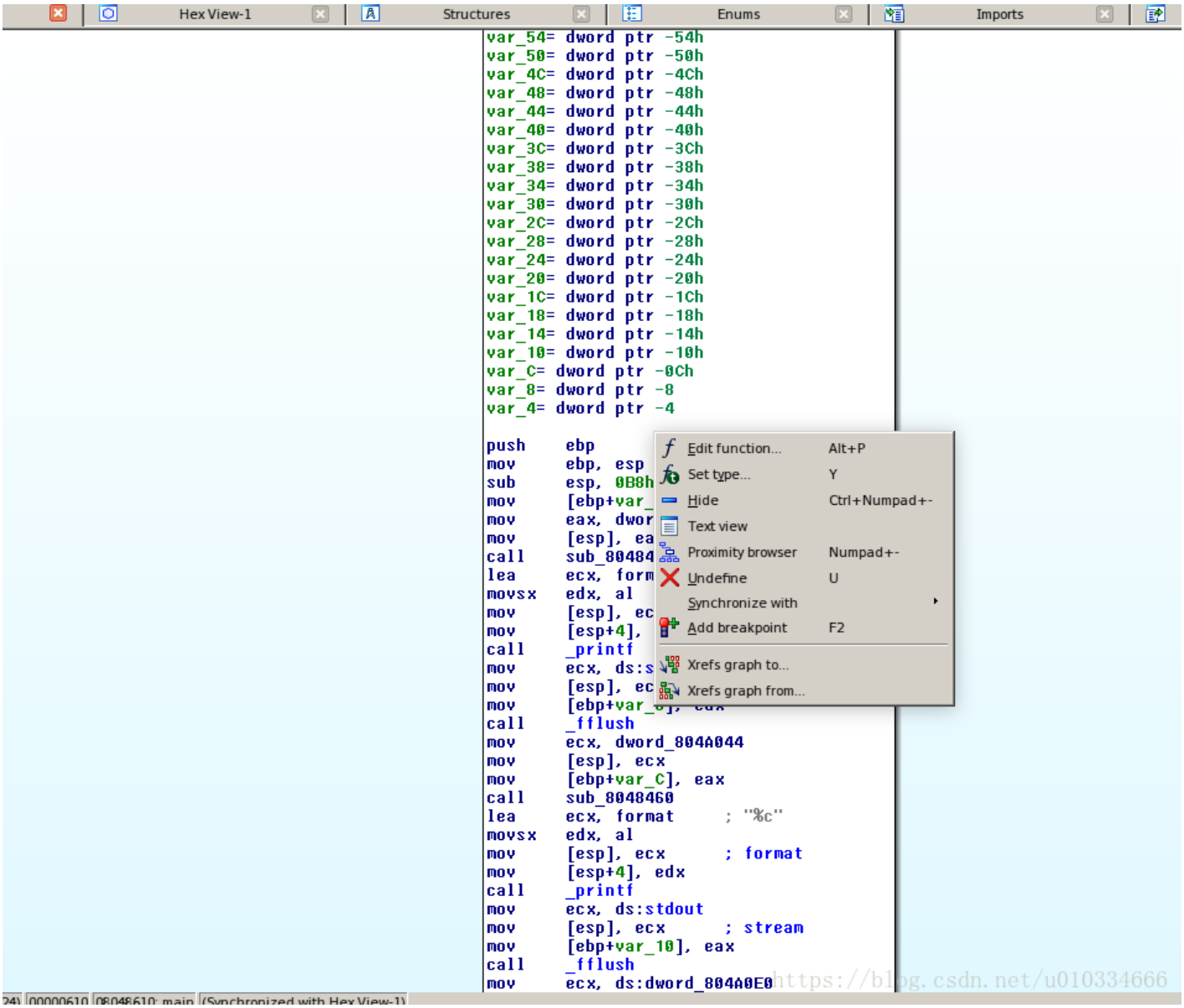
```
text:08048340                   assume cs:_text
text:08048340                   ;org 8048340h
text:08048340                   assume es:nothing, ss:nothing, ds:_data, fs:nothing, gs:nothing
text:08048340
text:08048340 ; =============== S U B R O U T I N E =======================================
text:08048340
text:08048340 ; Attributes: noreturn
text:08048340
text:08048340                   public start
text:08048340 start             proc near
text:08048340                   xor     ebp, ebp
text:08048342                   pop     esi
text:08048343                   mov     ecx, esp
text:08048345                   and     esp, 0FFFFFFF0h
text:08048348                   push    eax
text:08048349                   push    esp             ; stack_end
text:0804834A                   push    edx             ; rtld_fini
text:0804834B                   call    sub_8048373
text:08048350                   add     ebx, 1CB0h
text:08048356                   lea     eax, (nullsub_1 - 804A000h)[ebx]
text:0804835C                   push    eax             ; fini
text:0804835D                   lea     eax, (sub_8048AF0 - 804A000h)[ebx]
text:08048363                   push    eax             ; init
text:08048364                   push    ecx             ; ubp_av
text:08048365                   push    esi             ; argc
text:08048366                   mov     eax, offset main
text:0804836C                   push    eax             ; main
text:0804836D                   call    ___libc_start_main
text:08048372                   hlt
text:08048372 start             endp
text:08048372
text:08048373
```


，学过c的都知道，main是起始函数，双击main进入，按esc可以返回上一层

```
.text:08048610 var_8           = dword ptr -8
.text:08048610 var_4           = dword ptr -4
.text:08048610
.text:08048610                 push    ebp
.text:08048611                 mov     ebp, esp
.text:08048613                 sub     esp, 0B8h
.text:08048619                 mov     [ebp+var_4], 0
.text:08048620                 mov     eax, dword_804A03C
.text:08048625                 mov     [esp], eax
.text:08048628                 call    sub_8048460
.text:0804862D                 lea     ecx, format     ; "%c"
.text:08048633                 movsx   edx, al
.text:08048636                 mov     [esp], ecx      ; format
.text:08048639                 mov     [esp+4], edx
.text:0804863D                 call    _printf
.text:08048642                 mov     ecx, ds:stdout
.text:08048648                 mov     [esp], ecx      ; stream
.text:0804864B                 mov     [ebp+var_8], eax
.text:0804864E                 call    _fflush
.text:08048653                 mov     ecx, dword_804A044
.text:08048659                 mov     [esp], ecx
.text:0804865C                 mov     [ebp+var_C], eax
.text:0804865F                 call    sub_8048460
.text:08048664                 lea     ecx, format     ; "%c"
.text:0804866A                 movsx   edx, al
.text:0804866D                 mov     [esp], ecx      ; format
.text:08048670                 mov     [esp+4], edx
.text:08048674                 call    _printf
.text:08048679                 mov     ecx, ds:stdout
.text:0804867F                 mov     [esp], ecx      ; stream
.text:08048682                 mov     [ebp+var_10], eax
.text:08048685                 call    _fflush
.text:0804868A                 mov     ecx, ds:dword_804A0E0
.text:08048690                 mov     [esp], ecx
.text:08048693                 mov     [ebp+var_14], eax
.text:08048696                 call    sub_8048460
.text:0804869B                 lea     ecx, format     ; "%c"
.text:080486A1                 movsx   edx, al
.text:080486A4                 mov     [esp], ecx      ; format
.text:080486A7                 mov     [esp+4], edx
.text:080486AB                 call    _printf
.text:080486B0                 mov     ecx, ds:stdout
.text:080486B6                 mov     [esp], ecx      ; stream
.text:080486B9                 mov     [ebp+var_18], eax
.text:080486BC                 call    _fflush
.text:080486C1                 mov     ecx, dword_804A050
.text:080486C7                 mov     [esp], ecx
.text:080486CA                 mov     [ebp+var_1C], eax
.text:080486CD                 call    sub_8048460
.text:080486D2                 lea     ecx, format     ; "%c"
```


然后F5便可以显示c代码，有时候函数多的时候你连main函数都不知道在哪，re的题目通常都没加密字符串，所以我教你们另一种可行办法，若为图形界面

，



右键，text view，查看汇编代码，然后alt+T搜索可用的字符串，字符串你可以运行程序获得，也可以用工具获得，radare2也是一个神器，在此不多说，我也不怎么会用，搜索到后可以利用

```
text:08048610 ; Attributes: bp-based frame
text:08048610
text:08048610 ; int __cdecl main(int, char **, char **)
text:08048610 main            proc near               ; DATA XREF: start+26↑o
text:08048610
text:08048610 var_B0          = dword ptr -0B0h
text:08048610 var_AC          = dword ptr -0ACh
text:08048610 var_A8          = dword ptr -0A8h
text:08048610 var_A4          = dword ptr -0A4h
```


那个箭头进行回溯，找到真正的主要函数，此题没字符串，就不多讲，这里直接F5查看c代码，

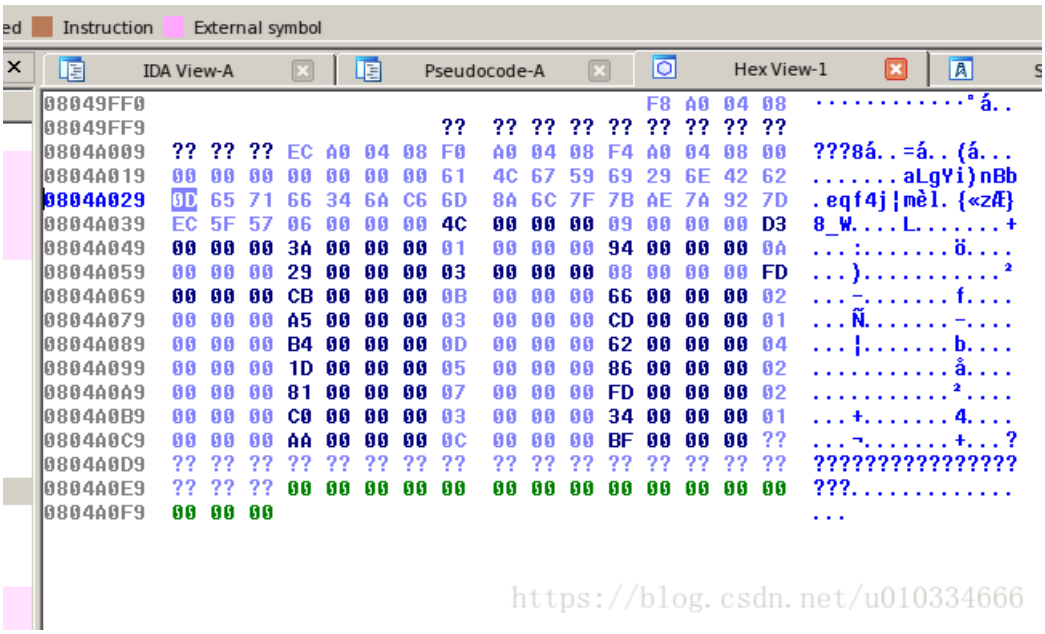
```
v0 = sub_8048460(dword_804A03C);
printf("%c", v0);
fflush(stdout);
v1 = sub_8048460(dword_804A044);
printf("%c", v1);
fflush(stdout);
v2 = sub_8048460(dword_804A0E0);
printf("%c", v2);
fflush(stdout);
v3 = sub_8048460(dword_804A050);
printf("%c", v3);
fflush(stdout);
v4 = sub_8048460(dword_804A058);
printf("%c", v4);
fflush(stdout);
v5 = sub_8048460(dword_804A0E4);
printf("%c", v5);
fflush(stdout);
v6 = sub_8048460(dword_804A064);
printf("%c", v6);
fflush(stdout);
v7 = sub_8048460(dword_804A0E8);
printf("%c", v7);
fflush(stdout);
v8 = sub_8048460(dword_804A070);
printf("%c", v8);
fflush(stdout);
v9 = sub_8048460(dword_804A078);
printf("%c", v9);
fflush(stdout);
v10 = sub_8048460(dword_804A080);
printf("%c", v10);
fflush(stdout);
v11 = sub_8048460(dword_804A088);
printf("%c", v11);
fflush(stdout);
v12 = sub_8048460(dword_804A090);
printf("%c", v12);
fflush(stdout);
v13 = sub_8048460(dword_804A098);
printf("%c", v13);
fflush(stdout);
v14 = sub_8048460(dword_804A0A0);
printf("%c", v14);
fflush(stdout);
```

char v3; // al@1

发觉一点有用的信息也没有，在此我已经不知道怎么做了，看了writeup后知道，这里传入的是flag的顺序，

```
.data:0804A020 byte_804A020    db 61h    ; DATA XREF: sub_8048460:loc_804848C↑r
.data:0804A021 byte_804A021    db 4Ch    ; DATA XREF: sub_8048460+33↑r
.data:0804A022 byte_804A022    db 67h    ; DATA XREF: sub_8048460:loc_80484A6↑r
.data:0804A023 byte_804A023    db 59h    ; DATA XREF: sub_8048460+4D↑r
.data:0804A024 byte_804A024    db 69h    ; DATA XREF: sub_8048460:loc_80484C0↑r
.data:0804A025 byte_804A025    db 29h    ; DATA XREF: sub_8048460+67↑r
.data:0804A026 byte_804A026    db 6Eh    ; DATA XREF: sub_8048460:loc_80484DA↑r
.data:0804A027 byte_804A027    db 42h    ; DATA XREF: sub_8048460+81↑r
.data:0804A028 byte_804A028    db 62h    ; DATA XREF: sub_8048460:loc_80484F4↑r
.data:0804A029 byte_804A029    db 0Dh    ; DATA XREF: sub_8048460+9B↑r
.data:0804A02A byte_804A02A    db 65h    ; DATA XREF: sub_8048460:loc_804850E↑r
.data:0804A02B byte_804A02B    db 71h    ; DATA XREF: sub_8048460+B5↑r
.data:0804A02C byte_804A02C    db 66h    ; DATA XREF: sub_8048460:loc_8048528↑r
.data:0804A02D byte_804A02D    db 34h    ; DATA XREF: sub_8048460+CF↑r
.data:0804A02E byte_804A02E    db 6Ah    ; DATA XREF: sub_8048460:loc_8048542↑r
.data:0804A02F byte_804A02F    db 0C6h   ; DATA XREF: sub_8048460+E9↑r
.data:0804A030 byte_804A030    db 6Dh    ; DATA XREF: sub_8048460:loc_804855C↑r
.data:0804A031 byte_804A031    db 8Ah    ; DATA XREF: sub_8048460+103↑r
.data:0804A032 byte_804A032    db 6Ch    ; DATA XREF: sub_8048460:loc_8048576↑r
.data:0804A033 byte_804A033    db 7Fh    ; DATA XREF: sub_8048460+11D↑r
.data:0804A034 byte_804A034    db 7Bh    ; DATA XREF: sub_8048460:loc_8048590↑r
.data:0804A035 byte_804A035    db 0AEh   ; DATA XREF: sub_8048460+137↑r
.data:0804A036 byte_804A036    db 7Ah    ; DATA XREF: sub_8048460:loc_80485AA↑r
.data:0804A037 byte_804A037    db 92h    ; DATA XREF: sub_8048460+151↑r
.data:0804A038 byte_804A038    db 7Dh    ; DATA XREF: sub_8048460:loc_80485C4↑r
.data:0804A039 byte_804A039    db 0ECh   ; DATA XREF: sub_8048460+16B↑r
.data:0804A03A byte_804A03A    db 5Fh    ; DATA XREF: sub_8048460:loc_80485DE↑r
.data:0804A03B byte_804A03B    db 57h    ; DATA XREF: sub_8048460+185↑r
.data:0804A03C dword_804A03C   dd 6      ; DATA XREF: main+10↑r
```


也就是这一串字符里含有flag，打开ida的hex窗口，hex是16进制



在小窗口里有的选，发觉一串可疑字符，里面可疑独处含有flag，顺序明显被打乱了，还有一堆乱码混淆我们，然后通过手动整理可以得到传入的参数为

list=[6,9,0,1,0xA,0,8,0,0xB,2,3,1,0xD,4,5,2,7,2,3,1,0xC]，

有未知参数，在.bss段，未初始化的全局变量，这里网上的writeup解释得都很模糊，我从c语言的学习中得知，全局变量未初始化便为0，所以未知的参数全填充为0,（个人理解），然后在那串字符中，找规律得到flag为奇数位的字符，偶数位的字符只是拿来加密的（在下找规律从小学开始就不好，恕我愚钝，没看出），然后就是得到奇数位字符，奇数位的数据你按r可以转换成字符

flag=['a','g','i','n','b','e','f','j','m','l','{','z','}','_']
偶数位字符

xor=[0x4c,0x59,0x29,0x42,0x0D,0x71,0x34,0xC6,0x8A,0x7F,0xAE,0x92,0xEC,0x57]

然后通过python脚本跑一下就出来了

```
#!/usr/bin/env python
# coding=utf-8
list=[6,9,0,1,0xA,0,8,0,0xB,2,3,1,0xD,4,5,2,7,2,3,1,0xC]
flag=['a','g','i','n','b','e','f','j','m','l','{','z','}','_']
xor=[0x4c,0x59,0x29,0x42,0x0D,0x71,0x34,0xC6,0x8A,0x7F,0xAE,0x92,0xEC,0x57]
result=""
for i in range(21):
    result+=flag[list[i]]
print result
```

题目下载：下载地址密码： j4fn