

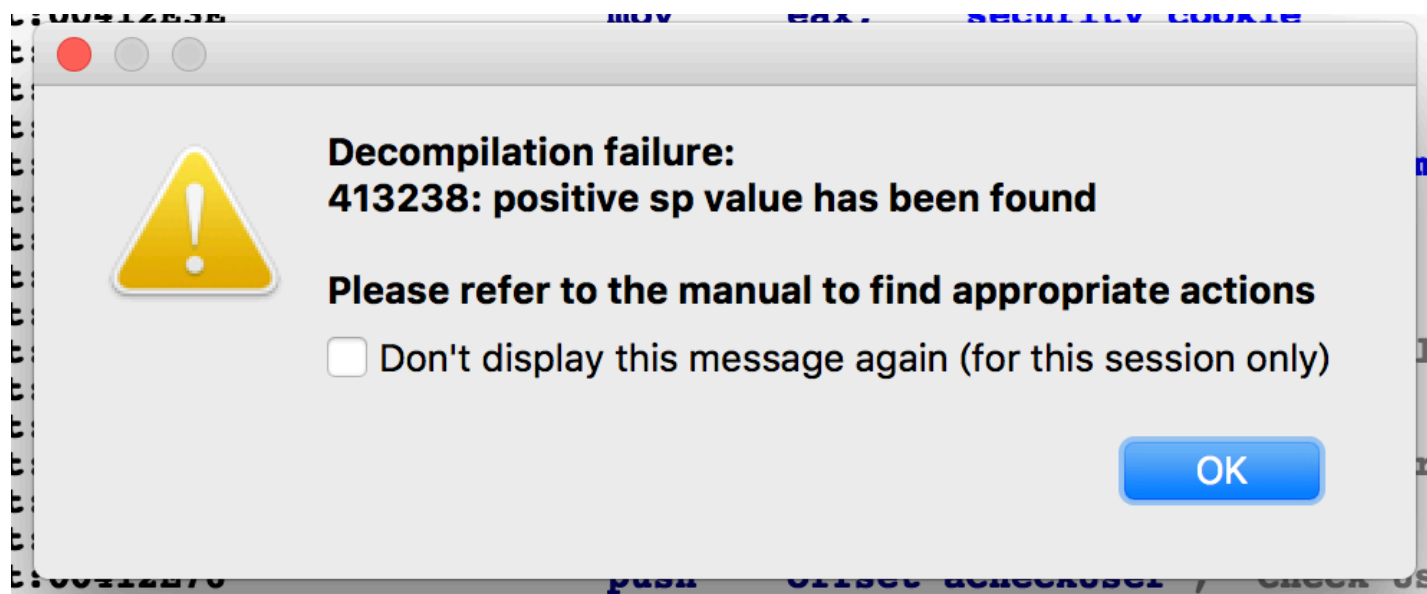
Evr

这个题主要学习的点在于 分析程序逻辑和如何从ida中 dump数组内容。

真正逆向工程： 几点必须记得： 1、程序都是人写的 区分人写的代码与编译器自己加上去的 区分出库与程序代码 不需要仔细逆向编译器代码 人的风格与机器差异很大 2、bin都是编译器生成的 程序生成是有规律的 exellib1llib2llib3 3、代码重用很普遍 str xref code style 开发自动识别工具 4、bin是可执行的 动态调试验证自己猜测 debugging tracing 符号执行 Taint analysis 5、有耐心！ 6、七分猜三分逆 7、一块一块的看代码 常用算法熟练掌握 常见数据结构 图 树 哈希表 常用设计模式 proxy stub etc 常见的框架

这道题 还是涉及到很多小技巧的。

首先我们找到main_0的函数 但是无法反编译，报错如下：



我们去这个地址的位置看一下：

```
.text:00413229      add     esp, 110h
.text:0041322F      cmp     ebp, esp
.text:00413231      call   j__RTC_CheckEsp
.text:00413236      mov     esp, ebp
.text:00413238      pop     ebp
.text:00413239      retn
.text:00413239      main_0      endp ; sp-analysis failed
.text:00413239
.text:00413239 ; -----
.text:0041323A      align 4
.text:0041323C      dword_41323C      dd 1 ; DATA XREF: _main_
.text:00413240      dd offset dword_413244
```

这个位置指针分析失败，接下来的操作：option->General->Disassembly

勾选 stack pointer

Disassembly
Analysis
Cross-references
Strings
Browser
Graph
Misc

Address representation

☐ Function offsets
☒ Include segment addresses
☒ Use segment names

Display disassembly line parts

☒ Line prefixes (non-graph)
☒ Stack pointer
☒ Comments
☒ Repeatable comments
☐ Auto comments

Number of opcode bytes (non-graph)

Display disassembly lines

☒ Empty lines
☒ Borders between data/code (non-graph)
☐ Basic block boundaries (non-graph)
☐ Source line numbers
☒ Try block lines

Instruction indentation (non-graph)

Comments indentation (non-graph)

Right margin (non-graph)

Spaces for tabulation

line prefix example: seg000:0FE4

这样我们就可以看到 栈指针的信息：

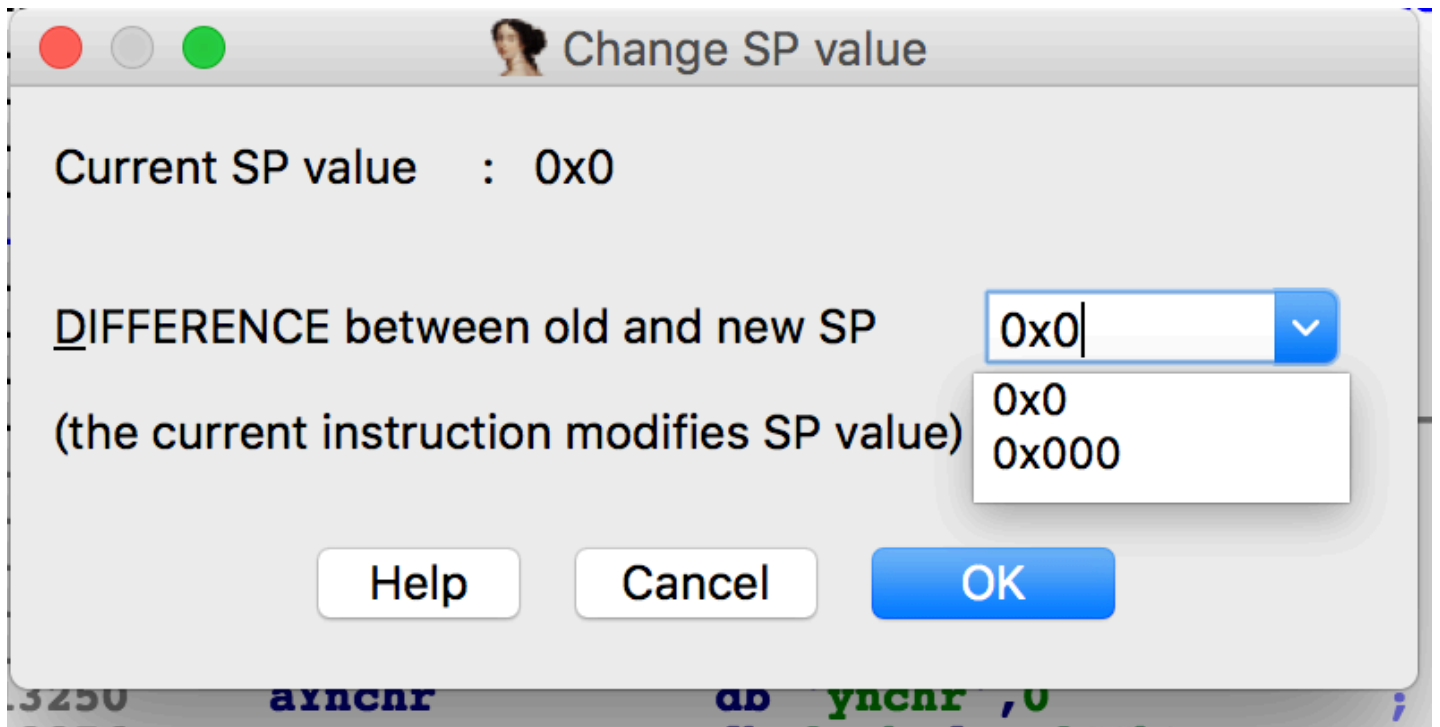
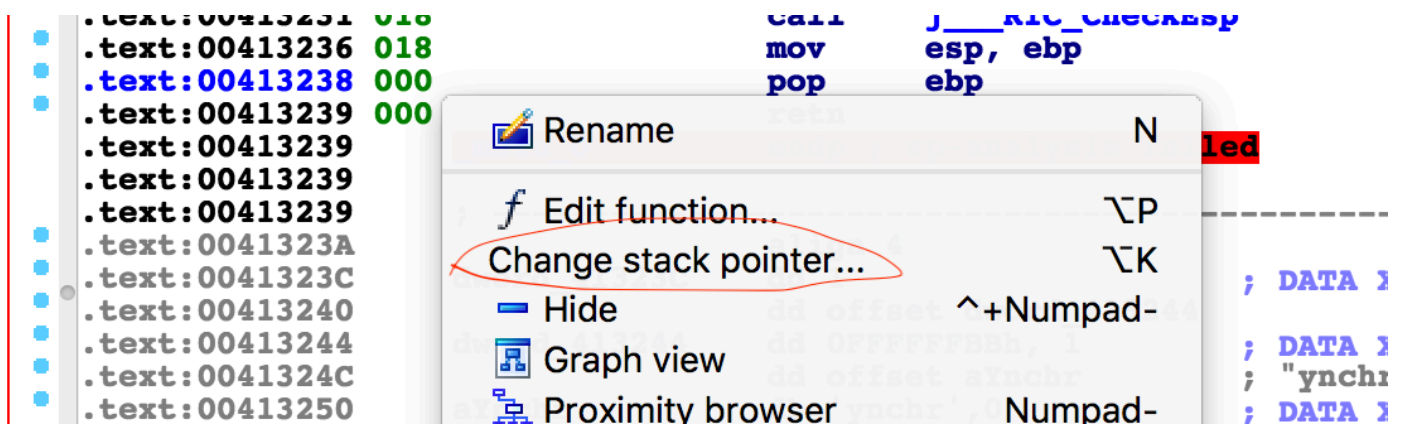
<pre> :0041321B 138 :0041321C 134 :0041321D 130 :0041321E 12C :0041321F 128 :00413222 128 :00413224 128 :00413229 128 :0041322F 018 :00413231 018 :00413236 018 :00413238 -F8 :00413239 -FC :00413239 :00413239 </pre>	<pre> pop edx pop edi pop esi pop ebx mov ecx, [ebp+var_4 xor ecx, ebp call j_@__security_ add esp, 110h cmp ebp, esp call j___RTC_CheckE mov esp, ebp pop ebp retn </pre>
--	--

main_0

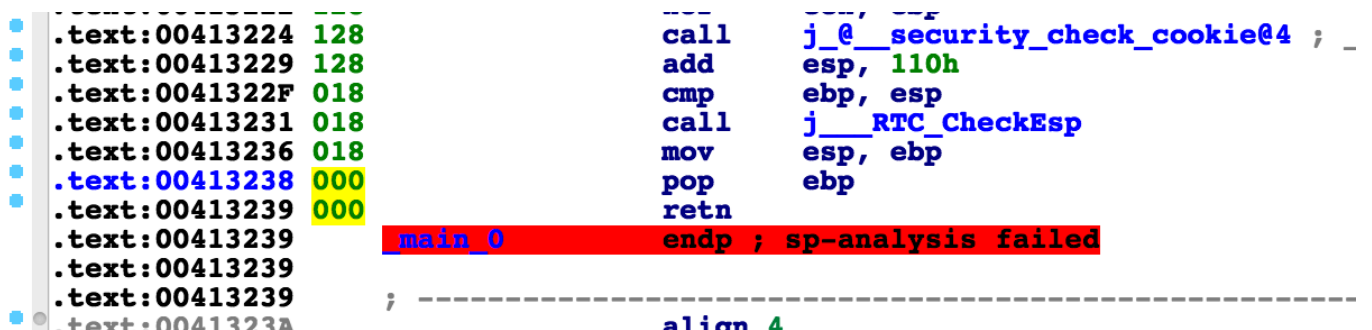
endp ; sp-analysis fail

现在需要把 238 239的位置改成 大于0的数字就可以，但是我们要从236开始改 改成和现实的数字相同就可以

快捷键alt+k 但是我的不行 可以右键选中



之后我们会得到一个这样的结果，然后 我们就可以 反编译生成伪代码了。关于这部分内容，以后再深究吧，现在见过的例子太少。也不是现在的重点。



~

```
__int64 main_0()
{
    int v0; // edx
```

```

__int64 v1; // ST08_8
char v3; // [esp-14h] [ebp-130h]
char *v4; // [esp-Ch] [ebp-128h]
signed int v5; // [esp-8h] [ebp-124h]
signed int v6; // [esp-4h] [ebp-120h]
char v7; // [esp+0h] [ebp-11Ch]
bool v8; // [esp+Fh] [ebp-10Dh]
char v9; // [esp+D7h] [ebp-45h]
int i; // [esp+E0h] [ebp-3Ch]
bool v11; // [esp+EFh] [ebp-2Dh]
bool v12; // [esp+FBh] [ebp-21h]
bool v13; // [esp+107h] [ebp-15h]
HMODULE v14; // [esp+110h] [ebp-Ch]

v14 = GetModuleHandleW(0);
sub_4113F7("Welcome to HCTF 2017\n\n", v7);
sub_4113F7("Mark.09 is hijacking Shinji Ikari now...\n\n", v7);
sub_4113F7("Check User: \n", v7);
v6 = 256;
sub_411154("%s", (unsigned int)&Str);
if ( !sub_411316() )
{
    sub_41114A();
    exit(0);
}
sub_4111F9();
sub_4113F7("Check Start Code: \n", v6);
v5 = 128;
v4 = input;
sub_411154("%s", input, 128); // 接收input
while ( getchar() != 10 )
;
if ( j_strlen(input) != 35 )
{
    sub_411398();
    sub_4110FF();
    exit(0);
}
sub_41114F(&input_1, input); // 变换1
sub_4110EB(sub_41105A, &sub_4111EA, dword_41B780, dword_41B784);
if ( sub_411361(1) )
{
    sub_411398();
    sub_4110FF();
    exit(0);
}
v8 = sub_4110EB(sub_411023, sub_41139D, dword_41B770, dword_41B774) != 0;
v13 = v8;

```

```

sub_411023(input_2, &input_1);          // 变换2
sub_411258(dword_41B770, dword_41B774, 204);
if ( sub_411361(2) )
{
    sub_411398();
    sub_4110FF();
    exit(0);
}
v8 = sub_4110EB(sub_41106E, &sub_411046, dword_41B778, dword_41B77C) != 0;
v12 = v8;
sub_41106E(input_3, &input_1);          // 变换3
sub_411258(dword_41B778, dword_41B77C, 205);
if ( sub_411361(3) )
{
    sub_411398();
    sub_4110FF();
    exit(0);
}
v8 = sub_4110EB(sub_41105A, &sub_4111EA, dword_41B780, dword_41B784) != 0;
v11 = v8;
sub_41105A(input_4, &input_1);          // 变换4
sub_411258(dword_41B780, dword_41B784, 221);
for ( i = 0; i < 7; ++i )
{
    byte_41B577[i] = input_2[i];
    byte_41B57E[i] = input_3[i];
    byte_41B585[i] = input_4[i];
}
if ( sub_411447(&input_1, &unk_41B0DC) )    // 校验
{
    MessageBoxA(0, "> DETONATION FUNCTION\n    READY", "WILLE", 0);
    sub_4113F7("[Y/N]?\\n", v3);
    sub_411154("%c", &v9, 1);
    if ( v9 != 89 && v9 != 121 )
    {
        sub_411398();
        sub_4110FF();
    }
    else
    {
        sub_411082();
        sub_4113F7("Prevent IMPACT success\\n", v3);
    }
}
else
{
    sub_411398();
    sub_4110FF();
}

```

```
    }  
    system("pause");  
    HIDWORD(v1) = v0;  
    LODWORD(v1) = 0;  
    return v1;  
}
```

上面是一个main_0函数 返汇编之后的伪代码。

我做了一下注释，看起来没有那么乱。

贴几个链接：<https://www.xctf.org.cn/library/details/3d4a7ca72000b89ff88fba1534697dd8c74f4d5f/>

<https://www.xctf.org.cn/library/details/70e83cb9f1950d046a7465e4c1de05b0dae5511d/>

在这就不描述这是怎样的一个过程了。

~

接下来说下 如何 dump数组：

将一段字符以数组的形式dump出来：先按“d”，调整数组元素的长度(db/dw/dd)，再按”*”生成数组，同时设置数组的大小，最后按”SHIFT + e”弹出dump框。

tips：

反汇编窗口将表达式中的变量改成数组：先按”Y”再输入 “char *”

F5反编译错误 用Alt+k

按y定义函数

“Shift + *” 导出数据

R 16进制转字符串

H 切换进制

exp:

```

flag_enc = [30, 21, 2, 16, 13, 72, 72, 111, 221, 221, 72, 100, 99, 215, 4
6, 44, 254, 106, 109, 42, 242, 111, 154, 77, 139, 75, 10, 138, 79, 69, 23,
70, 79, 20, 11]
flag=""
for i in range(7):
    flag += chr(flag_enc[i]^0x76)
for i in range(7):
    for j in range(32,127):
        a = j ^ 0x76 ^ 0xad
        a = (2 * a) & 0xaa | ((a & 0xaa) >> 1)
        if(a==flag_enc[i+7]):
            flag += chr(j)
for i in range(7):
    for j in range(32,127):
        a = j ^ 0x76 ^ 0xbe
        a = (4 * a) & 0xcc | ((a & 0xcc) >> 2)
        if(a==flag_enc[i+14]):
            flag += chr(j)
for i in range(7):
    for j in range(32,127):
        a = j ^ 0x76 ^ 0xef
        a = (16 * a) & 0xf0 | ((a & 0xf0) >> 4)
        if(a==flag_enc[i+21]):
            flag += chr(j)
for i in range(7):
    flag += chr(flag_enc[i+28]^0x76)
print flag

```

相关阅读:

为什么会出现指针分析失败: <https://blog.csdn.net/dj0379/article/details/8699219>

<https://www.xctf.org.cn/library/details/f48a9b2d65759cff3cf65b0a90c22a334b95e4d9/>

<https://www.xctf.org.cn/library/details/43e728d7475e32f0654081f441b2f2faa0924fcc/>