

南邮CTF逆向题第二道ReadAsm2解题思路

原创

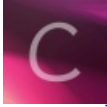
iqiqiya 于 2017-12-23 18:38:47 发布 2126 收藏 1

分类专栏: -----南邮CTF 我的CTF进阶之路 文章标签: CTF 南邮CTF writeup 逆向

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/xiangshangbashaonian/article/details/78881498>

版权



-----南邮CTF 同时被 2 个专栏收录

6 篇文章 0 订阅

订阅专栏

我的CTF进阶之路

108 篇文章 18 订阅

订阅专栏

首先看下提示

ReadAsm2

150

读汇编是逆向基本功。

给出的文件是func函数的汇编

main函数如下

输出的结果即为flag, 格式为 flag{*****}, 请连flag[]一起提交

编译环境为linux gcc x86-64

调用约定为System V AMD64 ABI

请不要利用汇编器, IDA等工具。。这里考的就是读汇编与推算汇编结果的能力

```
int main(int argc, char const *argv[])
{
    char input[] = {0x0, 0x67, 0x6e, 0x62, 0x63, 0x7e, 0x74, 0
x62, 0x69, 0x6d,
                    0x55, 0x6a, 0x7f, 0x60, 0x51, 0x66, 0x63, 0
x4e, 0x66, 0x7b,
                    0x71, 0x4a, 0x74, 0x76, 0x6b, 0x70, 0x79, 0
x66, 0x1c};
    func(input, 28);
    printf("%s\n", input+1);
    return 0;
}
```

参考资料:

<https://github.com/vefcos/reverse-engineering-for-beginners>

《汇编语言》王爽

《C反汇编与逆向分析技术揭秘》

2.asm

Key

SUBMIT

分析main函数 可以发现主要功能还是在于func函数中 于是我们仔细看func函数

```

1 0000000004004e6 <func>: ;4004e6—列表示该指令对应的虚拟内存地址 55—列为该指令对应的计算机指令
2 4004e6: 55 push rbp ;入栈,将寄存器的值压入调用 bp 栈中
3 4004e7: 48 89 e5 mov rbp,rs;建立新栈帧,别掉函数栈帧栈底地址放入寄存器
4 4004ea: 48 89 7d e8 mov QWORD PTR [rbp-0x18],rdi;对应main中input[] 这时i=0 // [rbp-0x18] = input[0]
5 4004ee: 89 75 e4 mov DWORD PTR [rbp-0x1c],esi;放入28 // [rbp-0x1c] = 28
6 4004f1: c7 45 fc 01 00 00 00 mov DWORD PTR [rbp-0x4],0x1;首先将0x1赋值给 [rbp-0x4] // i = 1
7 4004f8: eb 28 jmp 400522 <func+0x3c>;接着跳转到400522的位置 //for(i=1;i<=28;i++) 下面以第一次过程为例
8 4004fa: 8b 45 fc mov eax,DWORD PTR [rbp-0x4];将 [rbp-0x4] 的值赋给eax寄存器 //即令eax=i =1
9 4004fd: 48 63 d0 movsxd rdx,eax;将eax的值带符号扩展,并传送至rdx中 //即令rdx=eax =i =1
10 400500: 48 8b 45 e8 mov rax,QWORD PTR [rbp-0x18];将rax的值给input[0] //即令rax = input[0] = [rbp-0x18]
11 400504: 48 01 d0 add rax,rdx;将rdx的值加上rax再赋值给rax //即 rax=input[1] =i+input[0] =rdx+rax
12 400507: 8b 55 fc mov edx,DWORD PTR [rbp-0x4];将 [rbp-0x4] 的值给edx //即令edx=i =1
13 40050a: 48 63 ca movsxd rcx,edx;将edx的值带符号扩展,并传送至rcx中 //即令rcx=i =1
14 40050d: 48 8b 55 e8 mov rdx,QWORD PTR [rbp-0x18];将 [rbp-0x18] 的值给rdx //即令rdx=[rbp-0x18] =input[0]
15 400511: 48 01 ca add rdx,rcx;将rcx的值加上rdx再赋值给rdx //即i++ rdx=input[1]
16 400514: 0f b6 0a movzx ecx,BYTE PTR [rdx];将rdx无符号扩展,并传送至ecx //即ecx=chr(rdx) =chr(input[0])
17 400517: 8b 55 fc mov edx,DWORD PTR [rbp-0x4];edx = [rbp-0x4] //即edx=i =1
18 40051a: 31 ca xor edx,ecx;将edx与ecx异或 //i^input[0]
19 40051c: 88 10 mov BYTE PTR [rax],dl;rax = dl
20 40051e: 83 45 fc 01 add DWORD PTR [rbp-0x4],0x1; [rbp-0x4]++ //i++
21 400522: 8b 45 fc mov eax,DWORD PTR [rbp-0x4];将 [rbp-0x4] 的值赋给eax寄存器 //eax = i
22 400525: 3b 45 e4 cmpeax,DWORD PTR [rbp-0x1c];将 [rbp-0x1c] 中的值与eax值比较 第一次就是28
23 400528: 7e d0 jle 4004fa <func+0x14>;如果<=那么就跳到4004fa //if eax即i <=28跳到4004fa继续循环
24 40052a: 90 nop ;空指令
25 40052b: 5d pop rbp ;出栈
26 40052c: c3 ret ;ret相当于return

```

0000000004004e6<func>;4004e6—列表示该指令对应的虚拟内存地址 55—列为该指令对应的计算机指令

4004e6:55push rbp ;入栈,将寄存器的值压入调用 bp 栈中

4004e7:4889 e5 mov rbp,rs;建立新栈帧,别掉函数栈帧栈底地址放入寄存器

4004ea:48897d e8 movQWORDPTR[rbp-0x18],rdi;对应main中input[]这时i=0 // [rbp-0x18] = input[0]

4004ee:8975 e4 movDWORDPTR[rbp-0x1c],esi;放入28 // [rbp-0x1c] = 28

4004f1: c745 fc 01000000movDWORDPTR[rbp-0x4],0x1;首先将0x1赋值给 [rbp-0x4] // i = 1

4004f8: eb28jmp400522<func+0x3c>;接着跳转到400522的位置 //for(i=1;i<=28;i++) 下面以第一次过程为例

4004fa:8b45 fc moveax,DWORDPTR[rbp-0x4];将 [rbp-0x4] 的值赋给eax寄存器 //即令eax=i =1

4004fd:4863 d0 movsxd rdx,eax;将eax的值带符号扩展,并传送至rdx中 //即令rdx=eax =i =1

400500:488b45 e8 mov rax,QWORDPTR[rbp-0x18];将rax的值给input[0] //即令rax = input[0] = [rbp-0x18]

400504:4801 d0 add rax,rdx;将rdx的值加上rax再赋值给rax //即 rax=input[1] =i+input[0] =rdx+rax

400507:8b55 fc movedx,DWORDPTR[rbp-0x4];将 [rbp-0x4] 的值给edx //即令edx=i =1

40050a:4863 ca movsxd rcx,edx;将edx的值带符号扩展,并传送至rcx中 //即令rcx=i =1

40050d:488b55 e8 mov rdx,QWORDPTR[rbp-0x18];将 [rbp-0x18] 的值给rdx //即令rdx=[rbp-0x18] =input[0]

400511:4801 ca add rdx,rcx;将rcx的值加上rdx再赋值给rdx //即i++ rdx=input[1]

400514:0f b6 0amovzxecx,BYTEPTR[rdx];将rdx无符号扩展,并传送至ecx //即ecx=chr(rdx) =chr(input[0])

400517:8b55 fc movedx,DWORDPTR[rbp-0x4];edx = [rbp-0x4] //即edx=i =1

40051a:31 ca xoredx,ecx;将edx与ecx异或 //i^input[0]

40051c:8810movBYTEPTR[rax],dl;rax = dl

40051e:8345 fc 01addDWORDPTR[rbp-0x4],0x1; [rbp-0x4]++ //i++

400522:8b45 fc moveax,DWORDPTR[rbp-0x4];将 [rbp-0x4] 的值赋给eax寄存器 //eax = i

400525:3b45 e4 cmpeax,DWORDPTR[rbp-0x1c];将 [rbp-0x1c] 中的值与eax值比较 第一次就是28

400528:7e d0 jle4004fa<func+0x14>;如果<=那么就跳到4004fa //if eax即i <=28跳到4004fa继续循环

40052a:90nop;空指令

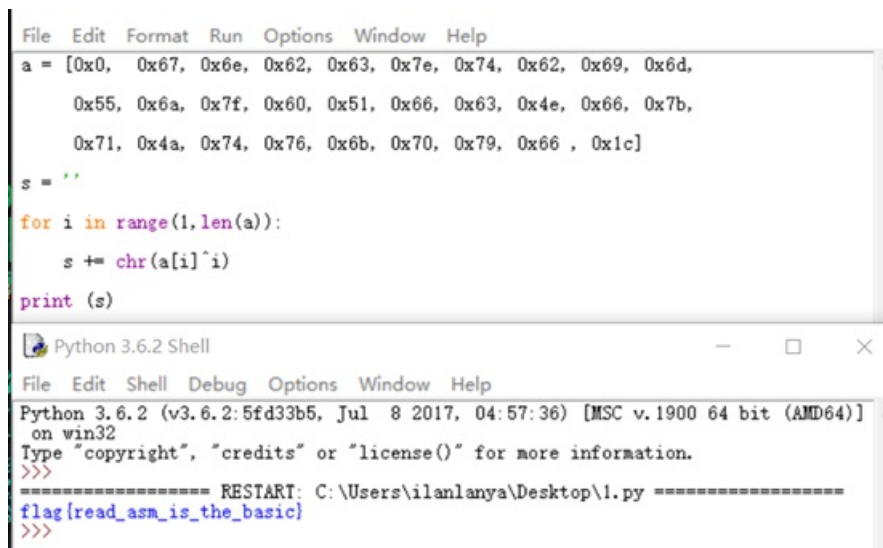
40052b:5dpop rbp ;出栈

40052c: c3ret;ret相当于return

分析可得Python3代码:

```
a = [0x0, 0x67, 0x6e, 0x62, 0x63, 0x7e, 0x74, 0x62, 0x69, 0x6d,  
0x55, 0x6a, 0x7f, 0x60, 0x51, 0x66, 0x63, 0x4e, 0x66, 0x7b,  
0x71, 0x4a, 0x74, 0x76, 0x6b, 0x70, 0x79, 0x66 , 0x1c]  
  
s = ''  
  
for i in range(1,len(a)):  
  
s += chr(a[i]^i)  
  
print (s)
```

运行结果如下:



```
File Edit Format Run Options Window Help  
a = [0x0, 0x67, 0x6e, 0x62, 0x63, 0x7e, 0x74, 0x62, 0x69, 0x6d,  
0x55, 0x6a, 0x7f, 0x60, 0x51, 0x66, 0x63, 0x4e, 0x66, 0x7b,  
0x71, 0x4a, 0x74, 0x76, 0x6b, 0x70, 0x79, 0x66 , 0x1c]  
  
s = ''  
  
for i in range(1,len(a)):  
s += chr(a[i]^i)  
  
print (s)  
  
Python 3.6.2 Shell  
File Edit Shell Debug Options Window Help  
Python 3.6.2 (v3.6.2:5fd33b5, Jul 8 2017, 04:57:36) [MSC v.1900 64 bit (AMD64)]  
on win32  
Type "copyright", "credits" or "license()" for more information.  
>>>  
===== RESTART: C:\Users\ilanlanya\Desktop\l.py =====  
flag{read_asm_is_the_basic}  
>>>
```

推荐看下这篇 指令讲解的很棒

https://www.cnblogs.com/Chesky/p/nuptzj_re_writeup.html

关于x86-64寄存器

http://www.360doc.com/content/14/0428/09/9408846_372826893.shtml