reverse-easyGo(新手向)

原创

Spwpun ● 于 2019-09-08 13:05:33 发布 ● 407 ☆ 收藏 分类专栏: writeup 逆向 版权声明:本文为博主原创文章,遵循 <u>CC 4.0 BY-SA</u> 版权协议,转载请附上原文出处链接和本声明。 本文链接: <u>https://blog.csdn.net/lplp9822/article/details/89416074</u> 版权



writeup 同时被 2 个专栏收录

15 篇文章 1 订阅

订阅专栏



5 篇文章 0 订阅 订阅专栏

2019信安国赛初赛reverse-easyGo

记录解题思路,不一定做得出来! 我现在能做了! (2019-09-08) ELF格式程序,利用kali试着运行一下:

root@kali:~/下載# ./easyGo Please input you flag like flag{123} to judge: flag{113} Try again! Come on!

应该只是一个简单的判断而已。 使用gdb看看有没有调试的信息,gdb之前没有用过,这里只能硬上弓了:

ali:~/下载# gdb -q easyGo Reading symbols from easyGo...(no debugging symbols found)...done. (gdb) b start No symbol table is loaded. Use the "file" command. Make breakpoint pending on future shared library load? (y or [n]) y Breakpoint 1 (start) pending. (gdb) file No executable file now. No symbol file now. (gdb) r Starting program: No executable file specified. Use the "file" or "exec-file" command. (gdb) file easyGo Reading symbols from easyGo...(no debugging symbols found)...done. https://blog.csdn.net/lplp9822 (gdb) q

是说因为没有发现调试符号(debug symbols),所以就直接完成了。 使用ida打开来看也只是一大堆函数符号,从字符串窗口也得不到任何信息,根据短时间内解出来的队伍数来看,我想这道题的 解法应该挺简单的。用OD? 行吗?试试。

之前写夭折了,菜鸡的成长之路真是曲折!

下面是详细的 writeup:

Go语言逆向,去掉了符号,导致很多函数都不能识别出来,然后使用网上的脚本可以对其进行处理, "https://raw.githubusercontent.com/strazzere/golang_loader_assist/master/golang_loader_assist.py",这里直接在 ida 中 File->Scritpt File(Alt + F7) 加载该脚本文件就可以识别了:

繴 IDA - E:\CTF题目\国赛ctf\reverse\easyGo.i64 (easyGo)								
<u>F</u> ile	<u>E</u> dit	<u>J</u> ump	Searc <u>h</u>	<u>V</u> iew	Deb <u>u</u> gger	<u>O</u> ptions	<u>W</u> indows	Help
	<u>N</u> ew in	stance						at .,† -
1	Open							
	<u>L</u> oad f	ile					•	ruction
	<u>P</u> roduc	ce file					►	
1	Script	<u>f</u> ile				Alt+	-F7	Pseudoco
₫	Script	co <u>m</u> mar	nd			Shif	t+F2	10000000
-	<u>S</u> ave					Ctrl	+W	10000000
	Sa <u>v</u> e a	s					1	10000000
6	Take d	atabase	snapshot	<u>t</u>		St//DIOCtrl	+Shift+W	0000000

然后看到的结果是这样的:

<u> </u>			
Function name	.text:0000000004952AA	mov	rdx, [rsp+18h]
f fmt ss hexString	.text:0000000004952AF	lea	rbx, [rsp+58h]
f fmt ss scanOne	.text:0000000004952B4	mov	[rsp], rbx
f fmt_errorHandler	.text:0000000004952B8	mov	[rsp+8], rdx
f fmt_ss_advance	.text:0000000004952BD	mov	[rsp+10h], rax
f fmt_ss_doScanf	.text:0000000004952C2	mov	[rsp+18h], rcx
<u>f</u> fmt_glob_func1	.text:0000000004952C7	call	runtime_slicebytetostring
<u>f</u> fmt_glob_func2	.text:0000000004952CC	mov	rax, cs:gword 572B00
f tmt_ss_loken_tunci	.text:0000000004952D3	mov	rcx, [rsp+20h]
f fmt_init_ializers	.text:0000000004952D8	mov	rdx, [rsp+28h]
f sub 494DB0	.text:0000000004952DD	mov	[rsp], rax
f sub 494E30	.text:0000000004952E1	mov	[rsp+8], rcx
f sub_494EE0	.text:0000000004952E6	mov	[rsp+10h], rdx
f sub_494F90	.text:0000000004952EB	call	encoding base64 Encoding DecodeString
f sub_495040	.text:0000000004952F0	mov	rax, [rsp+38h]
f_sub_4950C0	.text:0000000004952F5	mov	rcx, [rsp+30h]
1 main_main	.text:0000000004952FA	mov	rdx, [rsp+18h]
f main init ializers	.text:0000000004952FF	mov	rbx, [rsp+20h]
	.text:000000000495304	test	rcx, rcx
Line 1902 of 1904	000052C2 000000004052C2, main main	ul72 (Sunchronics	ad with New View-1)
	000552C2 000000004952C2: main_main	TTI/2 (Synchron126	https://blog.csdm.net/ipip8822

📃 Output window

Go 语言实际的主函数是 main_main,现在我看汇编还是看得挺懂的, Go 语言的参数传递也是通过栈的, 和一般的64位程序通过 寄存器传参是不一样的。

然后发现了这里:

.text:00000000004952C2	mov	[rsp+18h], rcx
.text:00000000004952C7	call	runtime_slicebytetostring
.text:00000000004952CC	mov	rax, cs:qword_572B00
.text:00000000004952D3	mov	rcx, [rsp+20h]
.text:00000000004952D8	mov	rdx, [rsp+28h]
.text:00000000004952DD	mov	[rsp], rax
.text:00000000004952E1	mov	[rsp+8], rcx
.text:00000000004952E6	mov	[rsp+10h], rdx
.text:00000000004952EB	call	<pre>encoding_base64Encoding_DecodeString</pre>
.text:00000000004952F0	mov	rax, [rsp+38h]
.text:00000000004952F5	mov	rcx, [rsp+30h]
.text:00000000004952FA	mov	rdx, [rsp+18h]
.text:00000000004952FF	mov	rbx, [rsp+20h]
.text:0000000000495304	test	rcx, rcx https://blog.csdn.net/lplp9822
±±.000000000000000	2	1 A0F44D

这里是在main函数中,通过调用这个函数解密程序中的真正的flag字符串,解密后就得到了flag。所以使用gdb调试在这里下断点 就可以了,不过需要注意要使用单步进入线程,不然可能会一直停在主函数开头。下面是详细的调试过程:

root@kali:~/文档# gdb easyGo GNU gdb (Debian 8.1-4) 8.1 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 easyGo...(no debugging symbols found)...done.
(gdb) b *0x4952eb //设置断点
Breakpoint 1 at 0x4952eb
(gdb) r //运行
Starting program: /root/文档/easyGo
[New LWP 51769]
[New LWP 51770]
[New LWP 51771]
[New LWP 51772]
Please input you flag like flag{123} to judge:
flag{
           //这里随便输入
Thread 1 "easyGo" hit Breakpoint 1, 0x00000000004952eb in ?? ()
                    //查看寄存器信息
(gdb) info reg
rax
              0xc00008a580 824634287488
              0x38 56
۰bx
rcx
              0xc00009c040 824634359872
              0x38 56
rdx
rsi
              0xc00009c000 824634359808
rdi
              0xc00009c040 824634359872
              0xc000086f88 0xc000086f88
              0xc000086e90 0xc000086e90
rsp
              0x1 1
r8
              AXA A
              0xc00009c040 824634359872
r10
r11
              0x0 0
              0xfffffffffffffff -1
°12
              0x2 2
r14
             0x1 1
r15
              0x80 128
              0x4952eb 0x4952eb
rip
eflags
              0x206 [ PF IF ]
              0x33 51
ds
              0x0 0
              0x0 0
fs
              0x0 0
---Type <return> to continue, or q <return> to quit---
              0x0 0
gs
(gdb) ni
            //单步步过
0x00000000004952f0 in ?? ()
(gdb) x/1s $rsi
                             //查看解密后将要对比的真正flag, x/1s 表示显示该地址处的1个字符串(string)
0xc00008c060: "flag{92094daf-33c9-431e-a85a-8bfbd5df98ad}"
(gdb)
```

这样就得到 flag 了,后面的程序逻辑就是将这个flag和你输入的flag对比,如果相等则输出提示 Congratulation...:

.text:000000000495318 cmp [rax+8], rbx text:000000000495310 iz short loc 495393 · 判断成功,跳到Congratulation			
text:000000000009531C jz short loc 495393 · 判断成功,跳到Congratulation	.text:0000000000495318	cmp	[rax+8], rbx
	.text:000000000049531C	jz	short loc_495393 ; 判断成功,跳到Congratulation

|--|

.text:0000000000495393 loc_495393:		; CODE XREF: main_main+1CC [†] j
.text:0000000000495393	mov	[rsp], rdx
.text:0000000000495397	mov	[rsp+8], rcx
.text:000000000049539C	mov	[rsp+10h], rbx
.text:00000000004953A1	call	runtime_memequal
.text:00000000004953A6	cmp	byte ptr [rsp+18h], 0
.text:0000000004953AB	jz	loc_49531E
.text:0000000004953B1	xorps	xmm0, xmm0
.text:0000000004953B4	movups	[rsp+100h+var_48], xmm0
.text:0000000004953BC	lea	rax, byte_4A6D00
.text:0000000004953C3	mov	qwor <u>d_ptr [rsp+100h+var_</u> 48], rax
.text:0000000004953CB	lea	rax, Congratulation
.text:00000000004953D2	mov	[rsp+0C0h], rax
.text:00000000004953DA	nop	
.text:0000000004953DB	mov	rax, cs:qword_572B18
.text:00000000004953E2	lea	rcx, off_4E28A0
.text:00000000004953E9	mov	[rsp], rcx
.text:00000000004953ED	mov	[rsp+8], rax
.text:0000000004953F2	lea	rax, [rsp+0B8h]
.text:00000000004953FA	mov	[rsp+10h], rax
.text:0000000004953FF	mov	qword ptr [rsp+18h], 1
.text:000000000495408	mov	qword ptr [rsp+20h], 1
.text:0000000000495411	call	fmt_Fprintln https://blog.csdn.net/lplp9