四叶草交流友谊赛

原创

 Gh0st
 ● ± 2020-05-08 10:24:44 发布
 ● 363
 ● 收藏

 分类专栏:
 re学习笔记

 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: <u>https://blog.csdn.net/qq_44115034/article/details/105759651</u>

版权



re学习笔记 专栏收录该内容

4 篇文章 1 订阅 订阅专栏 迷路的菜鸟,CTF从入门到放弃。

easyYM

运行程序。

Please input flag:123 flag:qwe123987 flag error!!! 请按任意键继续. . .

直接输出了flag? 其实并没有,题目描述也说了这不是真正的flag,真正的flag格式是flag{}。 拖进IDA,

1 ir	ntcdecl main_0()
2 {	
3	char v1; // [esp+4Ch] [ebp-20h]
4	
• 5	<pre>printf("Please input flag:");</pre>
6	<pre>scanf("%s", &v1);</pre>
• 7	<pre>printf("flag:qwe123987\n");</pre>
8	if (!strcmp(&v1, "qwe123987"))
9	<pre>printf("qwe123987 is not a flag!!!\n");</pre>
10	else
• 11	<pre>printf("flag error!!!\n");</pre>
• 12	<pre>system("pause");</pre>
• 13	return 0;
• 14}	

好像没什么有用信息。(太菜了,在这里还绕了好久) 看看题目描述,关键函数并不一定执行到了。

看看汇编代码。

.text:00401000		assume	cs:_text	
.text:00401000		;org 40	1000h	
.text:00401000		assume	es:nothing, ss:nothing, ds:_data, fs:nothing, gs:not	hing
.text:00401000		db 5 du	p(OCCh)	
.text:00401005	;			
.text:00401005		jmp	loc_401020 <	
.text:0040100A				
.text:0040100A	; ==========	=== S U B	R O U T I N E =================================	==
.text:0040100A				
.text:0040100A	; Attributes: t	thunk		
.text:0040100A				
.text:0040100A	; intcdecl m	nain(int a	argc, const char **argv, const char **envp)	
.text:0040100A	_main	proc ne	ar ; CODE XREF: start+E4↓p	
.text:0040100A		jmp	_main_0	
.text:0040100A	_main	endp		
.text:0040100A				
.text:0040100A	;			
.text:0040100F		align 2	ðh	
.text:00401020				
.text:00401020	<mark>loc_401020</mark> :		; CODE XREF: .text:00401005↑j	
.text:00401020		push	ebp	
.text:00401021		mov	ebp, esp	
.text:00401023		sub	esp, 48h	
.text:00401026		push	ebx	
.text:00401027		push	esi	
.text:00401028		push	edi	
.text:00401029		lea	edi, [ebp-48h]	
.text:0040102C		mov	ecx, 12h	
.text:00401031		mov	eax, 0CCCCCCCh	
.text:00401036		rep sto	sd	
.text:00401038		mov	dword ptr [ebp-4], 0	
.text:0040103F		jmp	short loc_40104A	
.text:00401041	;			
.text:00401041				
.text:00401041	loc_401041:		; CODE XREF: .text:00401071↓j	
.text:00401041		mov	eax, [ebp-4]	
1				

好吧,调用主函数之前还执行了下面这段代码,这里应该就是关键代码。在下面其实也发现了可疑字符串,更加明确了这点。

•	.text:00401044	add	eax, 1
•	.text:00401047	mov	[ebp-4], eax
	.text:0040104A		
	.text:0040104A loc_40104A:		; CODE XREF: .text:0040103F↑j
4.	.text:0040104A	push	offset aXozjx1vhz2uzc0 ; "XozjX1vHZ2uzc0TdS{qpZ3@{0U3nal7)dl2?"
•	.text:0040104F	call	_strlen
•	.text:00401054	add	esp, 4
•	.text:00401057	cmp	[ebp-4], eax
r =	.text:0040105A	jnb	short loc_401073

	.text:0040105C .text:0040105F	mov mov	ecx, [ebp-4] dl, byte ptr <mark>aXozjx1vhz2uzc0</mark> [ecx] ; "XozjX1vHZ2uzc0TdS{qpZ3@{OU3na17)d12?"
11	.text:00401065	xor	d1, 2
11	.text:00401068	mov	eax, [ebp-4]
1	.text:0040106B	mov	<pre>byte ptr aXozjx1vhz2uzc0[eax], dl ; "XozjX1vHZ2uzc0TdS{qpZ3@{0U3nal7)dl2?"</pre>
4	.text:00401071	jmp	short loc 401041
1	.text:00401073 ;		
1	.text:00401073		
1	.text:00401073 loc 401073:		; CODE XREF: .text:0040105A↑j
-	.text:00401073	mov	dword ptr [ebp-8], 0
-	.text:0040107A	jmp	short loc_401085
	.text:0040107C ;		
	.text:0040107C		
	.text:0040107C loc_40107C:		; CODE XREF: .text:004010AB↓j
+	.text:0040107C	mov	ecx, [ebp-8]
•	.text:0040107F	add	ecx, 1 +
•	.text:00401082	mov	[ebp-8], ecx
	.text:00401085		
	.text:00401085 loc_401085:		; CODE XREF: .text:0040107A↑j
4	.text:00401085	push	offset aRngcqgWqgCqg46 ; "Rngcqg]wqg]@cqg46]vm]fgap{rv]vm]egv]vjg"
	.text:0040108A	call	strlen
1	.text:0040108F	add	esp, 4
	.text:00401092	cmp	[ebp-8], eax
	.text:00401095	jnb	short loc_4010AD
1	.text:00401097	mov	edx, [ebp-8]
11	.text:0040109A	mov	al, byte ptr aRngcqgWqgCqg46[edx] ; "Rngcqg]wqg]@cqg46]vm]fgap{rv]vm]egv]vjg"
1	.text:004010A0	xor	al, 2 https://blog.csdn.net/ag_44115034
÷.		2 / 2 1	· 1 · · · · · · · · · · · · · · · · · ·

拖进OD调试,

☑ 文件(F) 查看(V) 调试(D) 插件(P) 选项(T) 窗口(W) 帮助(H) [+] 快捷菜单 1	Tools BreakPoint->				
1 emtwh	c ₽ k b r s ☵ 또 ?				
00401546 . A3 90E64200 nov dword ptr ds:[0x42E690],eax 00401546 . E8 F0400000 call easyYH.00405640 00401550 . A3 F0CC4200 nov dword ptr ds:[0x42CCFC],eax 00401550 . B3 F0CC4200 call easyYH.00405640 00401550 . E8 F030000 call easyYH.004045130 00401554 . E8 F030000 call easyYH.00404E0 00401554 . E8 S08000 call easyYH.00404E0 00401554 . E8 S08000 call easyYH.004020C0 00401554 . 8800 34CD420 nov ecx, dword ptr ds:[0x42CD34] 00401576 . 8900 38CD4220 nov dword ptr ds:[0x42CD34],ecx 00401576 . 8915 34CD4201 nov dword ptr ds:[0x42CD34] 00401576 . 52 nov dword ptr ds:[0x42CD34]					
00401577 . A1 20CD4200 now eax,dword ptr ds:[0x42CD2C] 00401570 . S8 . S800 28CD420 00401570 . S8600 28CD420 now ecx,dword ptr ds:[0x42CD28] 00401570 . S8600 28CD420 now ecx,dword ptr ds:[0x42CD28] 00401580 . S1 now ecx,dword ptr ds:[0x42CD28] 00401581 . E8 1FAFFFF	C 0 ES 0028 32(\overline{C}) 0(FFFFFFF) P 1 CS 0028 32(\overline{C}) 0(FFFFFFF) P 0 CS 0028 32(\overline{C}) 0(FFFFFFF) P 0 SS 0028 32(\overline{C}) 0(FFFFFFF) Z 0 DS 0028 32(\overline{C}) 0(FFFFFFF) Z 0 DS 0028 32(\overline{C}) 0(FFFFFFF) S 0 F 8 0053 32(\overline{C}) 209000(FFF) T 0 GS 0028 32(\overline{C}) 0(FFFFFFF) D 0 0 0 LastErr ERROR_SUCCESS (00000000) EFL 00000206 (N0,NB,NE,A,NS,PE,GE,G) V ST0 empty 0.0 ST1 empty 0.0 ST2 empty 0.0 ST3 empty 0.0				
	ST4 empty 0.0				
地理 10 10 10 10 10 10 10 10 10 10 10 10 10	▲?£? 0019FF36 02250EE8 00000001 0019FF36 02250EE8 0019FF48 084014n8 easyYH. <noduleentrypoint> 0019FF48 084014n8 easyYH.<noduleentrypoint> 0019FF48 084014n8 easyYH.<noduleentrypoint> 0019FF48 084014n8 easyYH.<noduleentrypoint> 0019FF88 0808080 0019FF54 080608080 0019FF54 080608080 0019FF54 080108080 0019FF54 08008080</noduleentrypoint></noduleentrypoint></noduleentrypoint></noduleentrypoint>				
M1 M2 M3 M4 M5 Command:	ESP				
VA: 00401584 -> 00401589 Size: (0x0005 - 00005 bytes) # ((0x0001 - 00001 dwords) 0ffset: 00001584 -> 00001589 Section (* Keasy 18) * * * * * * * * * * * * * * * * * * *				

我们在这里调用主函数的地方设置断点,修改一下汇编代码,调用之前的关键代码。

然后跟进,

🚼 吾爱破解 - easyYM.exe - [LCG - 主线程, 模块 - easyYM] — — — — — — — — — — — — — — — — — — —								
☑ 文件(F) 查看(V) 调试(D) 插件(P) 选项(T) 窗□(W) 帮助(H) [+] 快捷菜单	≜ Tools BreakPoint->							
· 普停 🔁 🗮 🗩 📕 🕨 📕 🔛 👘 1 em t w 1	877 ♥ ♥ ▼ ▶ II ♥ ♥ ♥ ↓ ♥ ↓ I e m t w h c P k b r s ☵ 또 ?							
0040104A > 68 <u>309A4200</u> <mark>} push</mark> easyYM.00429A30	ASCII 58,"ozjX1vHZ2uzc0TdS{qpZ3@{0U3nal7)d < 寄存器 (FPU)	<						
0040104F . E8 9C010000 <mark>call</mark> easyYM .004011F0	strlen EAX 0000000							
00401054 . 83C4 04 add esp,0x4	FCX 0000000							
00401057 . 3945 FC [local.1],eax	EDX ZEEEE5A							
0040105A 73 17 jnb short easyYM.00401073	FRX 00233000							
0040105C . 8B4D FC mov_ecx,[local.1]								
0040105F . 8A91 309A420 mov dl,byte ptr ds:[ecx+0x429A30]								
00401065 . 80F2 02 xor d1,0x2	EST 88481468 eacuVM /ModuleEntr	uPoint>						
00401068 . 8B45 FC mov_eax,[local.1]		gi orne,						
0040106B . 8890 309A420 nov bute ptr ds:[eax+0x429A30].dl								
00401071 .^ EB CE Limp short easyYM.00401041	EIP 6040106B easyYM.0040106B							
00401073 > C745 F8 0000 mov [local.2], 0x0								

0040107m cb. 99 Imp stort casyrr.s0401085 0040107C >	ASCII 52,"ngcqg]wqg]@cqgJ	P 1 CS 0023 32公 0(FFFFFFF) A 0 SS 0028 32位 0(FFFFFFF) 2 0 DS 0028 32位 0(FFFFFFF) S 0 FS 0053 32位 0(FFFFFFF) T 0 GS 0028 32位 0(FFFFFFF) D 0 0 0 LastErr ERROR_SUCCESS (00000000) EFL 00000206 (N0,NB,NE,A,NS,PE,GE,G)
8868189901 8482 58946280 mon al hute ntr dc-fedv+8v62045 dl=5A ('Z')	581	ST1 empty 0.0
ds:[00429A30]=58 ('X')		SI2 empty U.U
		ST3 empty 0.0
		STE omptu 0.0
地址 HEX 数据 F	ASCII	∧ 0019FEDC • 004014A0 easyYM. <moduleentrypoint></moduleentrypoint>
88481888 CC CC CC CC CC E9 16 88 88 89 21 81 88 88 CC	烫烫涕∎?≁?│	0019FEE0 · 004014A0 easyYM. <moduleentrypoint></moduleentrypoint>
	<i>쥷汤汤汤汤汤汤汤</i> 汤	0019FEE4 · 00233000
00401020 55 8B EC 83 EC 48 53 56 57 8D 7D B8 B9 12 00 00 Ū	U鄞泼HSUW唔腹∎	0019FEE8 · CCCCCCCC
00401030 00 B8 CC CC CC CC F3 AB C7 45 FC 00 00 00 00 EB .	.稆烫腆邩E??	0019FEEC · CCCCCCCC
00401040 09 8B 45 FC 83 C0 01 89 45 FC 68 <u>30 9A 42 00</u> E8 .	-婨鼉?塃黨❻欱.?│	0019FEF0 · CCCCCCCC
00401050 9C 01 00 00 83 C4 04 39 45 FC 73 17 8B 4D FC 8A 3	?兡 9E黶■婱鼕	0019FEF4 · CCCCCCCC
00401060 91 <u>30 9A 42 00</u> 80 F2 02 8B 45 FC 88 90 <u>30 9A 42</u> 3	?欱.■?婨鼒?欱、│	
Command:		ESF
VA: 0040106B -> 00401071 Size: (0x0006 - 00006 bytes)	# (0x0001 - 00001 dwords)	Offset: 0000106B -> 00001071 Section. <easytm> .text -</easytm>

随后发现在此处每次取出字符串中的一个字符与2异或,然后替换原来的字符串。设置断点跟踪查看即可得到替换后的字符串。



Tips	×
Please_use_Base64_to_decrypt_to_get_the_flag_Good_luck	
确定	

flag{I_L1ke_C++_Pr1mer~~~}

mwcc

运行程序。

2	×
迷雾重重???妈惹法克这	个怎么玩?

好像没什么有用信息,

但是细心将程序边框扩展开的话。



确定	
	取消 https://blog.csdn.nei/qq_44115034

我们就发现可以进行输入了,也有对应的输出。 我们随便输入一个flag,获取相关信息。



弹窗,Tips,flag error。 接下来静态,动态分析即可。 拖进IDA没有发现什么有用信息, 尝试动态分析。



跟进,并找到该函数的开始。

🔛 吾愛破解	f - mwcc.exe - [LCG - m	主线程, 模块 - mwcc]		-]	\times
C 文件(F)	查看(V) 调试(D) 插	牛(P) 选项(T) 窗囗(W) 帮助(H) [+] 快捷菜单 Too	ls BreakPoint->			-	ēх
暂停		▶ ▶ ▶ ▶ ↓ ▶ ↓ ▶ ↓ ↓ ↓ ↓ ↓ ↓	p k b r s 🖪 🔣 ?	☴ 🖪 🔲 🔘 🔛 🛄 吾 🗷 🗮 📕			
007A651D 007A651E 007A651F	CC CC CC	int3 int3 int3		 客存器(FPU) EAX 0019FFCG FCX 0019FFCG FCX 002723200 Bucc (ModulaEntruPaint) 	<	<	<
807A6528 007A6521 007A6523 007A6525 007A6525 007A652A 007A6538	<pre>-> 55 - 8BEC - 6A FF - 68 6647E300 - 64:A1 000000 - 50</pre>	push ebp mov ebp.esp push -0x1 push mwcc.00E34766 mov eax.dword ptr fs:[0] push eax		EDX 007733AD nwcc.(NoduleEntryPoint) EDX 00773AD nwcc.(NoduleEntryPoint) EBX 00241000 ESP 0019FF74 EBP 0019FF74 ESF 007733AD nwcc.(NoduleEntryPoint) ESI 007733AD nwcc.(NoduleEntryPoint)			
007A6531 007A6537 007A6538	- 81EC 3802000 - 53 - 56	sub esp.@x238 push ebx push esi	mwcc. <moduleentrypoint></moduleentrypoint>	EDI 00773380 nwcc.<(ModuleEntryPoint> EIP 007733AD nwcc. <moduleentrypoint> C 0 ES 002B 32位 0(FFFFFFF)</moduleentrypoint>			
007A6539 007A653A 007A653B 007A6541 007A6541	- 57 - 51 - 8DBD BCFDFFF - 89 8E000000 - 89 CCCCCCC	push edi push ecx lea edi,[local.145] mov ecx,0x8E pou eax @cccccccc	mwcc. <moduleentrypoint> mwcc.<moduleentrypoint></moduleentrypoint></moduleentrypoint>	P 1 CS 0023 32位 0(FFFFFFF) A 0 SS 0028 32位 0(FFFFFFF) 2 1 DS 0028 32位 0(FFFFFFF) 5 0 FS 0053 32位 244000(FFF)			
007A6540 007A654B 007A654D 007A654E 007A6553	- F3:AB - 59 - A1 34AFF400 - 33C5	rep stos dword ptr es:[edi] pop ecx nov eax,dword ptr ds:[0xF4AF34] xor eax,ebp	kerne132.76CD6359	T 0 GS 002B 32位 0(FFFFFFF) D 0 O 0 LastErr ERROR_SUCCESS (00000000 EFL 00000246 (N0,NB,E,BE,NS,PE,GE,LE)		
007A6555 007A6558 007A6559 007A6559	- 8945 F0 - 50 - 8045 F4	nov [local.4],eax push eax lea eax,[local.3]		ST0 empty 0.0 ST1 empty 0.0 ✓ ST2 empty 0.0 ST3 empty 0.0			
跳转来自	0076ADF2			ST4 empty 0.0 ST5 empty 0.0 ST6 empty 0.0			
084 01 000 084 01 000 094 01 02 0 094 01 02 0 094 01 03 0 094 01 04 0 094 01 05 0 094 01 05 0 094 01 07 0 094 01 08 0	HEX 数据 56 05 05 06 06 06 0 56 06 05 05 06 0 56 06 05 05 06 0 56 05 05 06 06 0 56 05 05 06 05 0 56 06 05 06 05 0 56 06 05 06 05 0 56 06 05 06 05 0	ASCII 0 00		 ▲ 0019FF74 76CD6359 ※回到 kernel32.76CD6359 Ø019FF78 Ø0241080 Ø019FF80 Ø019FF90 Ø0808080 Ø			Ŷ
MI M2 M3 VA: 007A65	ma mo i20 -> 007A6521	Command: 0000 - 55 000 - 00001 - 00001 bytes) # (0x1	0000 - 00000 dwords)	Offset: 00046920 -> 00046921 Section: <mwcc.< td=""><td>ESP I.net/qq.</td><td>2121010</td><td>NUNE 5034</td></mwcc.<>	ESP I.net/qq.	2121010	NUNE 5034

单步调试,在下图所示发现关键代码。



第一处将我们的输入压入堆栈,

第二处将类似flag的字符串压入堆栈。

第三处进行检验,

若输入正确,上面call的函数会将eax置1,输出this is flag。

若输出错误,上面call的函数会将eax置0,输出flag error。

CrackMe_clover01

```
这几道是Android逆向入门题,但作者之前没接触Android逆向。(菜醒。。)
```

大体思路其实都差不多,也是先查壳, 并没有加壳。

使用APKIDE或AndriodKiller, 查看MainActivity,直接找到flag



flag{cloveran}

CrackMe_clover02

运行程序, 随意输入一个用户名和注册码。

∩OX 夜神模拟器 6.6.0.5 And	ndroid 5	☑ ☎ = ₡ - □ ×
		💎 🖹 💈 9:24
程序未注册		
	Clover Sec 注册获取答 案	
username :	123	
sn : 123		
		注册
	无效用户名或注册码	

查找关键字

直接搜索运行程序时反馈的字符串 flag就写在下面。

<string name="unsuccessed">无效用户名或注册码</string> <string name="successed">恭喜您! 注册成功 F--21()a--g {CloverSec-android reverse}</string>

当然,查看smali代码修改关键的跳转指令也可。

flag{CloverSec-android reverse}