

DDCTF 2019 逆向题writeup(二)

原创

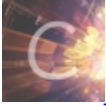
每昔 于 2019-04-13 12:00:58 发布 3164 收藏

分类专栏: [二进制漏洞](#) 文章标签: [DDCTF 2019 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/m0_37809075/article/details/89280350

版权



[二进制漏洞](#) 专栏收录该内容

11 篇文章 1 订阅

订阅专栏

文章目录

总结

脱壳分析

定位关键call

逆向分析

编写脚本

总结

- 其实输入只能是0-9和A-F有很大的提示作用, 开始没想明白
- and和add没看清, 走了很多弯路
- python模块itertools的排列组合函数
- 结合IDA分析会简便, 但是太烂不想脱壳dump

脱壳分析

首先是一个ASPack 2.12 -> Alexey Solodovnikov. 利用ESP定律脱壳。

定位关键call

这个call执行完后, 显示了input code:的字符串。所以需要跟入。

012A1580	- 0000 70202000	mov	ecx, dword ptr [0x12A2070]	MSVC70_11111111
012A15A0	- 8901	mov	dword ptr [ecx], eax	
012A15B0	- FF35 64302A00	push	dword ptr [0x12A3064]	
012A15B6	- FF35 68302A00	push	dword ptr [0x12A3068]	
012A15BC	- FF35 60302A00	push	dword ptr [0x12A3060]	
012A15C2	- E8 59FDFFFF	call	012A1320	
012A15C7	- 83C4 0C	add	esp, 0xC	
012A15CA	- A3 78302A01	mov	dword ptr [0x12A3078], eax	
012A15CF	- 391D 6C302A00	cmp	dword ptr [0x12A306C], ebx	
012A15D5	- 75 37	jnz	short 012A160E	
012A15D7	- 50	push	eax	[status
012A15D8	- FF15 A8202A00	call	dword ptr [0x12A20A8]	exit

跟进之后, 发现了明文字符串。其中call 012A11F0函数是对输入进行验证, 必须是0-9, A-F。其实这里就可以想到16进制数。

在这个函数里面兜兜转转好长时间，脑子瓦特了。

下一个关键的函数就是012A1240，需要进去看一下。

```
012A1374 . 68 24212A01 push 012A2124
012A1379 . FFD7 call edi
012A137B . 8D9424 24040 lea edx, dword ptr [esp+0x424]
012A1382 . 52 push edx
012A1383 . 68 30212A01 push 012A2130
012A1388 . FF15 B0202A00 call dword ptr [0x12A20B0]
012A138E . 83C4 24 add esp, 0x24
012A1391 . 8DB424 08040 lea esi, dword ptr [esp+0x408]
012A1398 . E8 53FEFFFF call 012A11F0
012A139D . 84C0 test al, al
012A139F . 75 12 jnz short 012A13B3
012A13A1 . 68 34212A01 push 012A2134
012A13A6 . FFD7 call edi
012A13A8 . 83C4 04 add esp, 0x4
012A13AB . 6A 00 push 0x0
012A13AD . FF15 A8202A00 call dword ptr [0x12A20A8]
012A13B3 . 8D8424 08080 lea eax, dword ptr [esp+0x808]
012A13BA . 50 push eax
012A13BB . 8DB424 0C040 lea esi, dword ptr [esp+0x40C]
012A13C2 . E8 79FEFFFF call 012A1240
012A13C7 . 68 FF030000 push 0x3FF
```

input code:
printf
%s
scanf
invalid input\n
status = 0x0
exit
n = 3FF (1023.)

反复调试，发现12A1000比较奇怪，因为它的输入是我们输入的flag。因此里面必有蹊跷。

```
012A12E0 . 99 mov eax, eax
012A12EE . 99 cdq
012A12EF . 2BC2 sub eax, edx
012A12F1 . D1F8 sar eax, 1
012A12F3 . 55 push ebp
012A12F4 . 50 push eax
012A12F5 . 8D4C24 14 lea ecx, dword ptr [esp+0x14]
012A12F9 . E8 02FDFFFF call 012A1000
012A12FE . 8B8C24 14040 mov ecx, dword ptr [esp+0x414]
012A1305 . 83C4 08 add esp, 0x8
012A1308 . 5F pop edi
012A1309 . 5D pop ebp
012A130A . 33CC xor ecx, esp
012A130C . E8 4D010000 call 012A145E
```

进去之后会发现里面有几个循环，循环不可怕，可怕的是循环里面有各种左移，右移，异或等操作。所以抓住这个函数死不松手进行分析。这个循环的输入是6个字符串为一组。输出是4个字符串。刚开始试的时候，发现输入如果不是6的倍数，看到输出会有“=”，一度以为是自定义base64。

```
012A1058 . 007D 00000000 lea eax, dword ptr [eax]
012A1060 . 8A0F mov cl, byte ptr [edi]
012A1062 . 884C34 14 mov byte ptr [esp+esi+0x14], cl
012A1066 . 8A5C24 15 mov bl, byte ptr [esp+0x15]
012A106A . 46 inc esi
012A106B . 4D dec ebp
012A106C . 47 inc edi
012A106D . 83FE 03 cmp esi, 0x3
012A1070 . 75 61 jnz short 012A10D3
012A1072 . 8A4424 14 mov al, byte ptr [esp+0x14]
012A1076 . 8AD0 mov dl, al
012A1078 . C0EA 02 shr dl, 0x2
012A107B . 24 03 and al, 0x3
012A107D . C0E0 04 shl al, 0x4
012A1080 . 885424 18 mov byte ptr [esp+0x18], dl
012A1084 . 8ACB mov cl, bl
012A1086 . C0E9 04 shr cl, 0x4
012A1089 . 02C1 add al, cl
012A108B . 884424 19 mov byte ptr [esp+0x19], al
012A108F . 8A4424 16 mov al, byte ptr [esp+0x16]
012A1093 . 8AD3 mov dl, bl
012A1095 . 80E2 0F and dl, 0xF
012A1098 . 02D2 add dl, dl
012A109A . 8AC8 mov cl, al
012A109C . 02D2 add dl, dl
```

逆向分析

重点对这里分析，会将输入拆分为6个一组。比如第一组123456。会再拆分为0x12，0x34，0x56。其中0x12经过计算放置在esp+0x18，0x34经过计算放置在esp+0x19处，以此类推。这里就理解为什么要求输入必须是16进制数里面的字符了。不是的话这里转化不了呀。。。

```
012A1062 . 884C34 14 mov byte ptr [esp+esi+0x14], c1
012A1066 . 8A5C24 15 mov bl, byte ptr [esp+0x15]
012A106A . 46 inc esi
012A106B . 4D dec ebp
012A106C . 47 inc edi
012A106D . 83FE 03 cmp esi, 0x3
012A1070 . 75 61 jnz short 012A10D3
012A1072 . 8A4424 14 mov al, byte ptr [esp+0x14]
012A1076 . 8AD0 mov dl, al
012A1078 . C0EA 02 shr dl, 0x2
012A107B . 24 03 and al, 0x3
012A107D . C0E0 04 shl al, 0x4
012A1080 . 885424 18 mov byte ptr [esp+0x18], dl
012A1084 . 8ACB mov cl, bl
012A1086 . C0E9 04 shr cl, 0x4
012A1089 . 02C1 add al, cl
012A108B . 884424 19 mov byte ptr [esp+0x19], al
012A108F . 8A4424 16 mov al, byte ptr [esp+0x16]
012A1093 . 8AD3 mov dl, bl
012A1095 . 80E2 0F and dl, 0xF
012A1098 . 02D2 add dl, dl
012A109A . 8AC8 mov cl, al
012A109C . 02D2 add dl, dl
012A109E . C0E9 06 shr cl, 0x6
012A10A1 . 02D1 add dl, cl
012A10A3 . 24 3F and al, 0x3F
012A10A5 . 885424 1A mov byte ptr [esp+0x1A], dl
012A10A9 . 884424 1B mov byte ptr [esp+0x1B], al
```

然后4个输出的数据就是edx，作为偏移，然后加上0x12A3020。这里的代码和第一道逆向题的思路差不多，就是索引到指定的字符串然后再与0x76异或。最后和谁去比呢？

```
012A10B0 > 0FB65434 18 movzx edx, byte ptr [esp+esi+0x18]
012A10B5 . 8A82 20302A0 mov al, byte ptr [edx+0x12A3020]
012A10BB . 34 76 xor al, 0x76
012A10BD . 0FB6C8 movzx ecx, al
012A10C0 . 51 push ecx
012A10C1 . 8D4C24 24 lea ecx, dword ptr [esp+0x24]
012A10C5 . FF15 3C202A0 call dword ptr [0x12A203C]
012A10CB . 46 inc esi
012A10CC . 83FE 04 cmp esi, 0x4
012A10CF . 7C DF jl short 012A10B0
012A10D1 . 33F6 xor esi, esi
012A10D3 > 85ED test ebp, ebp
012A10D5 . 75 89 jnz short 012A1060
012A10D7 . 85F6 test esi, esi
012A10D9 . 8E84 A2000000 je 012A1181
```

012A3020	37 34 35 32 33 30 31 3E 3F 3C 3D 3A 3B 38 39 26	7452301>?<=:;89&
012A3030	27 24 25 22 23 20 21 2E 2F 2C 17 14 15 12 13 10	'\$%"# !./,#####
012A3040	11 1E 1F 1C 1D 1A 1B 18 19 06 07 04 05 02 03 00	##### Y~.
012A3050	01 0E 0F 0C 46 47 44 45 42 43 40 41 4E 4F 5D 59	#####.FGDEBC@ANO]Y
012A3060	01 00 00 00 F8 1B 33 00 B8 2E 33 00 00 00 00 00	#####?3.?3.....
012A3070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
012A3080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
012A3090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

在之前调试这个位置看到DDCTF{reverse+}，其实就是和reverse+进行对比。其中reverse+长度是8。所以输入的长度是12的16进制字符。

```
012A13D3 . C64424 18 00 mov byte ptr [esp+0x18], 0x0
012A13D8 . E8 8D080000 call 012A1C6A
012A13DD . 8D9424 18080 lea edx, dword ptr [esp+0x818]
012A13E4 . 52 push edx
012A13E5 . 8D4424 1C lea eax, dword ptr [esp+0x1C]
012A13EE . 68 44212A01 push 012A2144
012A13EF . 50 push eax
```

012A13E8	- FF15 B420200	CALL	DWORD PTR [0x12A20B4]	! sprinter
012A13F5	- 83C4 1C	ADD	ESP, 0x1C	
012A13F8	- B9 50212A01	MOV	ECX, 012A2150	DDCTF{reverse+}
012A13FD	- 8D4424 08	LEA	EAX, DWORD PTR [ESP+0x8]	

编写脚本

```
import itertools

cover = [0x37,0x34,0x35,0x32,0x33,0x30,0x31,0x3E,0x3F,0x3C,0x3D,0x3A,0x3B,0x38,0x39,0x26,0x27,0x24,0x25,0x22,0x23,0x20,0x21,0x2E,0x2F,0x2C,0x17,0x14,0x15,0x12,0x13,0x10,0x11,0x1E,0x1F,0x1C,0x1D,0x1A,0x1B,0x18,0x19,0x06,0x07,0x04,0x05,0x02,0x03,0x00,0x01,0x0E,0x0F,0x0C,0x46,0x47,0x44,0x45,0x42,0x43,0x40,0x41,0x4E,0x4F,0x5D,0x59,0x01,0x00,0x00,0x00,0xF8,0x1B,0x2B,0x00,0xB8,0x2E,0x2B,0x00]
cc = ""

input_ = list(itertools.product([1,2,3,4,5,6,7,8,9,"A","B","C","D","E","F"],repeat = 6))

for i in range(len(input_)):

    a = int(str(input_[i][0])+str(input_[i][1]),16)
    one = a>>0x2

    b = int(str(input_[i][2])+str(input_[i][3]),16)
    tmp = (a&0x3)<<0x4
    tmp_1 = b>>0x4
    two = tmp_1+tmp

    c = int(str(input_[i][4])+str(input_[i][5]),16)
    tmp_2 = ((b&0xF)*2)*2
    tmp_3 = c>>0x6
    three = tmp_2+tmp_3

    four = c&0x3F

    fuzz = chr(cover[one]^0x76) + chr(cover[two]^0x76) + chr(cover[three]^0x76) + chr(cover[four]^0x76)
    if(fuzz == "reve" or fuzz == "rse+"):
        print str(hex(a))+str(hex(b))+str(hex(c))
```