[逆向][writeup]0ctf2015 r0ops

转载

weixin_33696822 于 2017-02-25 11:54:00 发布 53 收藏
原文链接：http://www.cnblogs.com/gsharpsh00ter/p/6441433.html
版权

这道逆向题相对其分值来说，实在是太难了，想要逆向这个程序，需要足够的耐心，而且还需要有一定的数学知识。

这是一个64位的ELF程序。首先用IDA进行分析，开始部分的逻辑是比较简单的：

```
1 void *__usercall handlemsg@<rax>(__int64 a1@<rbp>)
2 {
3   *(_DWORD *)(a1 - 4) = accept(3, 0LL, 0LL);
4   recv(*(_DWORD *)(a1 - 4), &qword_E0B10C0, 0x1000uLL, 0);
5   close(*(_DWORD *)(a1 - 4));
6   qmemcpy(&unk_E0AF0A0, &unk_E0B00A0, 0x1000uLL);
7   return &unk_E0AF8A0;
8 }
```

程序打开了一个套接字（描述符为3），然后监听这个套接字，在接收到消息后，会进行拷贝操作，然后返回。

最关键的就是返回操作。动态调试发现函数返回时，ret指令会取出栈上地址并跳转到该地址处执行一系列指令，结束后会再次返回到栈上，重复下一次过程。大概如下：

```
.text:000000000DEAD1F4 jmp     short loc_DEAD1F8
.text:000000000DEAD1F8 pop     rcx
.text:000000000DEAD1F9 retn
.text:000000000DEAD271 jmp     short loc_DEAD275
.text:000000000DEAD275 pop     r9
.text:000000000DEAD277 retn
```

换句话说，整个stack就是由一系列的ROP gadgets构成的ROPChain。在进行动态调试时，发现gadgets中含有大量的垃圾指令，比如无效的赋值，重复的拷贝等等，从而增大了我们理解程序的难度。

栈上的内容我们是可以获得的，想到可以写一个脚本，结合代码段与栈上的内容来还原这个ROPChain，但是后来发现gadgets中有不少rsp操作和条件跳转操作，这些操作的结果取决于动态执行的内容，用脚本来进行静态还原相当于写一个解释器了，难度可想而知，最终放弃。

耐着性子一步步跟踪gadget的执行，去掉无效的干扰指令，记录下有效指令，及相关内存、寄存器的值，如下：

```
 1 .text:000000000DEAD1F8 pop     rcx               ;0x0000000000000008
 2 .text:000000000DEAD275 pop     r9               ;0x1337DEADBEEF0095
 3 .text:000000000DEAD127 mov     rax, [rdi]       ;rax = 8bytes user input
 4 .text:000000000DEAD208 mov     [rsi], rax       ;[rsi] = 8bytes user input
 5 .text:000000000DEAD26B mov     r8, [rsi]       ;ri = 8bytes user input
 6 .text:000000000DEAD107 add     rdi, 8           ;rdi points to next 8 bytes user input
 7 .text:000000000DEAD27E mov     [rsi], r9       ;[rsi] = 0x1337DEADBEEF0095
 8 .text:000000000DEAD212 mov     rax, [rsi]       ;rax = 0x1337DEADBEEF0095
 9 .text:000000000DEAD1F0 pop     rbx               ;rbx = 0x000000000000CAFE
10 .text:000000000DEAD145 imul    rax, rbx          ;rax = 0x1337DEADBEEF0095 * 0x000000000000CAFE
11 .text:000000000DEAD208 mov     [rsi], rax
12 .text:000000000DEAD288 mov     r9, [rsi]       ;r9 = 0x1337DEADBEEF0095 * 0x000000000000CAFE
```

```
 13 .text:000000000DEAD1F0 pop     rbx                ;rbx = 0x000000000000BEEF
 14 .text:000000000DEAD131 add     rax, rbx           ;rax = 0x1337DEADBEEF0095 * 0x000000000000CAFE +
0x000000000000BEEF
 15 .text:000000000DEAD208 mov     [rsi], rax
 16 .text:000000000DEAD288 mov     r9, [rsi]        ;r9 = 0x1337DEADBEEF0095 * 0x000000000000CAFE +
0x000000000000BEEF
 17 .text:000000000DEAD2CC pop     r12                ;r12 = 0x0000000000000001
 18 .text:000000000DEAD292 pop     r10                ;r10 = 0x0000000000003419 = 13337
 19 .text:000000000DEAD1E8 pop     rax                ;rax = 0x0000000000000000
 20 .text:000000000DEAD1F0 pop     rbx                ;rbx = 0x0000000000000000
 21 .text:000000000DEAD200 pop     rdx                ;rdx = 0x00000000000001D8
 22 .text:000000000DEAD19F cmp     rax, rbx           ;
 23 .text:000000000DEAD1A2 jnz     short locret_DEAD1A7    ;not jmp
 24 .text:000000000DEAD1A4 add     rsp, rdx           ;rsp = 0x000000000E0AFBD0
 25 .text:000000000DEAD29B mov     [rsi], r10
 26 .text:000000000DEAD212 mov     rax, [rsi]       ;rax =  0x0000000000003419
 27 .text:000000000DEAD1F0 pop     rbx                ;0x0000000000000000
 28 .text:000000000DEAD200 pop     rdx                ;0xFFFFFFFFFFFFFDE0
 29 .text:000000000DEAD1AE cmp     rax, rbx           ;
 30 .text:000000000DEAD1B1 jz      short locret_DEAD1B6
 31 .text:000000000DEAD1B3 add     rsp, rdx           ;rsp = 0x000000000E0AF9F8
 32 .text:000000000DEAD212 mov     rax, [rsi]       ;rax = r10 = 0x0000000000003419
 33 .text:000000000DEAD1F0 pop     rbx                ;0x0000000000000001
 34 .text:000000000DEAD177 and     rax, rbx           ;rax = rax & rbx = 0x0000000000000001
 35 .text:000000000DEAD208 mov     [rsi], rax       ;[rsi] = 0x0000000000000001
 36 .text:000000000DEAD212 mov     rax, [rsi]       ;rax = 0x0000000000000001
 37 .text:000000000DEAD1F0 pop     rbx                ;0x0000000000000001
 38 .text:000000000DEAD200 pop     rdx                ;0x0000000000000068
 39 .text:000000000DEAD1AE cmp     rax, rbx           ;
 40 .text:000000000DEAD1B1 jz      short locret_DEAD1B6    ;jmped
 41 .text:000000000DEAD1B6 retn
 42 .text:000000000DEAD2D5 mov     [rsi], r12       ;[rsi] = 0x0000000000000001
 43 .text:000000000DEAD212 mov     rax, [rsi]       ;rax = 0x0000000000000001
 44 .text:000000000DEAD261 mov     [rsi], r8        ;[rsi] = input8
 45 text:000000000DEAD226 mov     rbx, [rsi]       ;rbx = input8
 46 .text:000000000DEAD145 imul    rax, rbx           ;rax = 0x0000000000000001 * input8
 47 .text:000000000DEAD208 mov     [rsi], rax       ;rax = input8
 48 .text:000000000DEAD2DF mov     r12, [rsi]       ;r12 = input8        r12用于保存幂的值
 49 .text:000000000DEAD261 mov     [rsi], r8        ;[rsi] = input8
 50 .text:000000000DEAD212 mov     rax, [rsi]       ;rax = input8
 51 .text:000000000DEAD226 mov     rbx, [rsi]       ;rbx = input8
 52 .text:000000000DEAD145 imul    rax, rbx           ;rax = input8*input*
 53 .text:000000000DEAD208 mov     [rsi], rax
 54 .text:000000000DEAD26B mov     r8, [rsi]        ;r8 = input8*input8
 55 .text:000000000DEAD29B mov     [rsi], r10       ;*rsi = 0x0000000000003419
 56 .text:000000000DEAD212 mov     rax, [rsi]       ;rax = 0x0000000000003419
 57 .text:000000000DEAD195 shr     rax, 1           ;rax = 0x0000000000001A0C
 58 .text:000000000DEAD208 mov     [rsi], rax       ;*rsi = 0x0000000000001A0C
 59 .text:000000000DEAD2A5 mov     r10, [rsi]       ;r10 = 0x0000000000001A0C
 60 .text:000000000DEAD1F0 pop     rbx                ;0x0000000000000000
 61 .text:000000000DEAD200 pop     rdx                ;0xFFFFFFFFFFFFFDE0
 62 .text:000000000DEAD1AE cmp     rax, rbx
 63 .text:000000000DEAD1B1 jz      short locret_DEAD1B6                        ;if r10/2 > 0
 64 .text:000000000DEAD1B3 add     rsp, rdx           ;0x000000000E0AF9F8
 65 .text:000000000DEAD1F0 pop     rbx                ;0x0000000000000001
 66 .text:000000000DEAD177 and     rax, rbx           ;rax = 0x0000000000000000
 67 .text:000000000DEAD208 mov     [rsi], rax       ;*rsi = 0x0000000000000000
 68 .text:000000000DEAD1F0 pop     rbx                ;0x0000000000000001
 69 .text:000000000DEAD200 pop     rdx                ;0x0000000000000068
```

```
 70 .text:000000000DEAD1AE cmp     rax, rbx
 71 .text:000000000DEAD1B1 jz      short locret_DEAD1B6    ;not jmp
 72 .text:000000000DEAD1B3 add     rsp, rdx            ;rsp = 0x000000000E0AFB20
 73 .text:000000000DEAD261 mov     [rsi], r8        ;*rsi = input8*input8
 74 .text:000000000DEAD212 mov     rax, [rsi]        ;rax = input8*input8
 75 .text:000000000DEAD226 mov     rbx, [rsi]        ;rbx = input8*input8
 76 .text:000000000DEAD145 imul    rax, rbx            ;rax = input8*input8*input8*input8
 77 .text:000000000DEAD208 mov     [rsi], rax        ;*rsi = input8*input8*input8*input8
 78 .text:000000000DEAD26B mov     r8, [rsi]        ;r8 = input8*input8*input8*input8
 79 .text:000000000DEAD29B mov     [rsi], r10        ;*rsi = 0x0000000000001A0C
 80 .text:000000000DEAD212 mov     rax, [rsi]        ;rax = 0x0000000000001A0C
 81 .text:000000000DEAD195 shr     rax, 1            ;0x0000000000000D06
 82 .text:000000000DEAD208 mov     [rsi], rax        ;*rsi = 0x0000000000000D06
 83 .text:000000000DEAD2A5 mov     r10, [rsi]        ;r10  = 0x0000000000000D06
 84 .text:000000000DEAD1F0 pop     rbx               ;rbx = 0x0000000000000000
 85 .text:000000000DEAD200 pop     rdx               ;rdx = 0x0000000000000068
 86 .text:000000000DEAD1AE cmp     rax, rbx
 87 .text:000000000DEAD1B1 jz      short locret_DEAD1B6;not jmp
 88 .text:000000000DEAD1B3 add     rsp, rdx            ;0x000000000E0AF9F8
 89 .text:000000000DEAD212 mov     rax, [rsi]        ;
 90 .text:000000000DEAD1F0 pop     rbx               ;0x0000000000000001
 91 .text:000000000DEAD177 and     rax, rbx            ;0x0000000000000000
 92 .text:000000000DEAD208 mov     [rsi], rax        ;*rsi=0x0000000000000000
 93 .text:000000000DEAD212 mov     rax, [rsi]        ;rax = r11 = 0x0000000000000000
 94 .text:000000000DEAD1F0 pop     rbx               ;0x0000000000000001
 95 .text:000000000DEAD200 pop     rdx               ;0x0000000000000068
 96 .text:000000000DEAD1AE cmp     rax, rbx
 97 .text:000000000DEAD1B1 jz      short locret_DEAD1B6 ; not jmp
 98 .text:000000000DEAD1B3 add     rsp, rdx            ;0x000000000E0AFB20
 99 .text:000000000DEAD261 mov     [rsi], r8        ;*rsi = input8*input8*input8*input8
100 .text:000000000DEAD212 mov     rax, [rsi]        ;rax = input8*input8*input8*input8
101 .text:000000000DEAD226 mov     rbx, [rsi]        ;rbx = input8*input8*input8*input8
102 .text:000000000DEAD145 imul    rax, rbx            ;rax =
input8*input8*input8*input8*input8*input8*input8*input8
103 .text:000000000DEAD208 mov     [rsi], rax
104 .text:000000000DEAD26B mov     r8, [rsi]        ;r8 =
input8*input8*input8*input8*input8*input8*input8*input8
105 .text:000000000DEAD29B mov     [rsi], r10        ;0x0000000000000D06
106 .text:000000000DEAD212 mov     rax, [rsi]        ;rax = 0x0000000000000D06
107 .text:000000000DEAD195 shr     rax, 1            ;rax = 0x0000000000000683
108 .text:000000000DEAD208 mov     [rsi], rax        ;*rsi = 0x0000000000000683
109 .text:000000000DEAD2A5 mov     r10, [rsi]        ;r10 = 0x0000000000000683
110 .text:000000000DEAD212 mov     rax, [rsi]        ;rax = 0x0000000000000683
111 .text:000000000DEAD1F0 pop     rbx               ;0x0000000000000000
112 .text:000000000DEAD200 pop     rdx               ;0xFFFFFFFFFFFFFDE0
113 .text:000000000DEAD1AE cmp     rax, rbx
114 .text:000000000DEAD1B1 jz      short locret_DEAD1B6                        ;if r10/2 > 0
115 .text:000000000DEAD1B3 add     rsp, rdx            ;0x000000000E0AF9F8
116 .text:000000000DEAD212 mov     rax, [rsi]        ;rax = 0x0000000000000683
117 .text:000000000DEAD1F0 pop     rbx               ;0x0000000000000001
118 .text:000000000DEAD177 and     rax, rbx            ;rax = 0x0000000000000001
119 .text:000000000DEAD208 mov     [rsi], rax        ;*rsi = 0x0000000000000001
120 .text:000000000DEAD212 mov     rax, [rsi]
121 .text:000000000DEAD1F0 pop     rbx               ;0x0000000000000001
122 .text:000000000DEAD200 pop     rdx               ;0x0000000000000068
123 .text:000000000DEAD1AE cmp     rax, rbx
124 .text:000000000DEAD1B1 jz      short locret_DEAD1B6;jmped
125 .text:000000000DEAD1B6 retn
126 .text:000000000DEAD2D5 mov     [rsi], r12        ;*rsi = input8
127 .text:000000000DEAD212 mov     rax, [rsi]        ;rax = input8
```

```
127 .text:000000000DEAD212 mov     rax, [rsi]          ;rax = input8
128 .text:000000000DEAD261 mov     [rsi], r8          ;*rsi =
input8*input8*input8*input8*input8*input8*input8*input8
129 .text:000000000DEAD226 mov     rbx, [rsi]          ;rbx =
input8*input8*input8*input8*input8*input8*input8*input8
130 .text:000000000DEAD145 imul    rax, rbx             ;rax = input(9)
131 .text:000000000DEAD208 mov     [rsi], rax
132 .text:000000000DEAD2DF mov     r12, [rsi]          ;r12 = r12 * r8 = input(9)
133 .text:000000000DEAD261 mov     [rsi], r8          ;*rsi = input(8)
134 .text:000000000DEAD212 mov     rax, [rsi]
135 .text:000000000DEAD226 mov     rbx, [rsi]
136 .text:000000000DEAD145 imul    rax, rbx
137 .text:000000000DEAD208 mov     [rsi], rax
138 .text:000000000DEAD26B mov     r8, [rsi]
139 .text:000000000DEAD29B mov     [rsi], r10
140 .text:000000000DEAD212 mov     rax, [rsi]
141 .text:000000000DEAD195 shr     rax, 1
142 .text:000000000DEAD208 mov     [rsi], rax
143 .text:000000000DEAD2A5 mov     r10, [rsi]
144 .text:000000000DEAD29B mov     [rsi], r10
145 .text:000000000DEAD212 mov     rax, [rsi]
146 .text:000000000DEAD1F0 pop     rbx                 ; 0x0000000000000000
147 ……
```

程序似乎没有结束的尽头，而且我们能够明显的感觉程序执行进入了一个循环。通过这些已经执行的代码，我们来大概分析一下程序的执行逻辑，并用伪码进行表示。如下：

```
; rcx = 8，应该是一个8次的循环
.text:000000000DEAD1F8 pop     rcx                 ;0x0000000000000008

;r8 = input[rcx], r9 = 0x1337DEADBEEF0095
.text:000000000DEAD275 pop     r9                 ;0x1337DEADBEEF0095
.text:000000000DEAD127 mov     rax, [rdi]         ;rax = 8bytes user input
.text:000000000DEAD208 mov     [rsi], rax         ;[rsi] = 8bytes user input
.text:000000000DEAD26B mov     r8, [rsi]         ;ri = 8bytes user input
.text:000000000DEAD107 add     rdi, 8             ;rdi points to next 8 bytes user input

;r9 = r9 * 0x000000000000CAFE + 0x000000000000BEEF
.text:000000000DEAD27E mov     [rsi], r9         ;[rsi] = 0x1337DEADBEEF0095
.text:000000000DEAD212 mov     rax, [rsi]         ;rax = 0x1337DEADBEEF0095
.text:000000000DEAD1F0 pop     rbx                ;rbx = 0x000000000000CAFE
.text:000000000DEAD145 imul    rax, rbx           ;rax = 0x1337DEADBEEF0095 * 0x000000000000CAFE
.text:000000000DEAD208 mov     [rsi], rax
.text:000000000DEAD288 mov     r9, [rsi]         ;r9 = 0x1337DEADBEEF0095 * 0x000000000000CAFE
.text:000000000DEAD1F0 pop     rbx                ;rbx = 0x000000000000BEEF
.text:000000000DEAD131 add     rax, rbx           ;rax = 0x1337DEADBEEF0095 * 0x000000000000CAFE +
0x000000000000BEEF
.text:000000000DEAD208 mov     [rsi], rax
.text:000000000DEAD288 mov     r9, [rsi]         ;r9 = 0x1337DEADBEEF0095 * 0x000000000000CAFE +
0x000000000000BEEF

;r12 = 1, r10 = 13337
.text:000000000DEAD2CC pop     r12                ;r12 = 0x0000000000000001
.text:000000000DEAD292 pop     r10                ;r10 = 0x0000000000003419 = 13337

.text:000000000DEAD1E8 pop     rax                ;rax = 0x0000000000000000
.text:000000000DEAD1F0 pop     rbx                ;rbx = 0x0000000000000000
.text:000000000DEAD200 pop     rdx                ;rdx = 0x00000000000001D8
.text:000000000DEAD19F cmp     rax, rbx           ;
```

```
.text:000000000DEAD1A2 jnz     short locret_DEAD1A7    ;not jmp
.text:000000000DEAD1A4 add     rsp, rdx            ;rsp = 0x000000000E0AFBD0
.text:000000000DEAD29B mov     [rsi], r10
.text:000000000DEAD212 mov     rax, [rsi]        ;rax =  0x0000000000003419
.text:000000000DEAD1F0 pop     rbx                ;0x0000000000000000
.text:000000000DEAD200 pop     rdx                ;0xFFFFFFFFFFFFFDE0
.text:000000000DEAD1AE cmp     rax, rbx            ;
.text:000000000DEAD1B1 jz      short locret_DEAD1B6
.text:000000000DEAD1B3 add     rsp, rdx            ;rsp = 0x000000000E0AF9F8

if (r10 & 0x01 == 0x01)
{
    r12 = r12 * r8;
    r8 = r8 * r8;
}

.text:000000000DEAD212 mov     rax, [rsi]        ;rax = r10 = 0x0000000000003419
.text:000000000DEAD1F0 pop     rbx                ;0x0000000000000001
.text:000000000DEAD177 and     rax, rbx            ;rax = rax & rbx = 0x0000000000000001
.text:000000000DEAD208 mov     [rsi], rax        ;[rsi] = 0x0000000000000001
.text:000000000DEAD212 mov     rax, [rsi]        ;rax = 0x0000000000000001
.text:000000000DEAD1F0 pop     rbx                ;0x0000000000000001
.text:000000000DEAD200 pop     rdx                ;0x0000000000000068
.text:000000000DEAD1AE cmp     rax, rbx            ;
.text:000000000DEAD1B1 jz      short locret_DEAD1B6    ;jmped
.text:000000000DEAD1B6 retn
.text:000000000DEAD2D5 mov     [rsi], r12        ;[rsi] = 0x0000000000000001
.text:000000000DEAD212 mov     rax, [rsi]        ;rax = 0x0000000000000001
.text:000000000DEAD261 mov     [rsi], r8        ;[rsi] = input8
text:000000000DEAD226 mov     rbx, [rsi]        ;rbx = input8
.text:000000000DEAD145 imul    rax, rbx            ;rax = 0x0000000000000001 * input8
.text:000000000DEAD208 mov     [rsi], rax        ;rax = input8
.text:000000000DEAD2DF mov     r12, [rsi]        ;r12 = input8        r12用于保存幂的值
.text:000000000DEAD261 mov     [rsi], r8        ;[rsi] = input8
.text:000000000DEAD212 mov     rax, [rsi]        ;rax = input8
.text:000000000DEAD226 mov     rbx, [rsi]        ;rbx = input8
.text:000000000DEAD145 imul    rax, rbx            ;rax = input8*input*
.text:000000000DEAD208 mov     [rsi], rax
.text:000000000DEAD26B mov     r8, [rsi]        ;r8 = input8*input8

r10 = r10 / 2;

.text:000000000DEAD29B mov     [rsi], r10        ;*rsi = 0x0000000000003419
.text:000000000DEAD212 mov     rax, [rsi]        ;rax = 0x0000000000003419
.text:000000000DEAD195 shr     rax, 1            ;rax = 0x0000000000001A0C
.text:000000000DEAD208 mov     [rsi], rax        ;*rsi = 0x0000000000001A0C
.text:000000000DEAD2A5 mov     r10, [rsi]        ;r10 = 0x0000000000001A0C

if (r10 != 0)
.text:000000000DEAD1F0 pop     rbx                ;0x0000000000000000
.text:000000000DEAD200 pop     rdx                ;0xFFFFFFFFFFFFFDE0
.text:000000000DEAD1AE cmp     rax, rbx
.text:000000000DEAD1B1 jz      short locret_DEAD1B6                        ;if r10/2 > 0

.text:000000000DEAD1B3 add     rsp, rdx            ;0x000000000E0AF9F8
.text:000000000DEAD1F0 pop     rbx                ;0x0000000000000001

if (r10 & 0x01 == 0)
{
    r8 = r8 * r8;
```

```
    r8 = r8*r8;
}

.text:000000000DEAD177 and     rax, rbx            ;rax = 0x0000000000000000
.text:000000000DEAD208 mov     [rsi], rax        ;*rsi = 0x0000000000000000
.text:000000000DEAD1F0 pop     rbx                ;0x0000000000000001
.text:000000000DEAD200 pop     rdx                ;0x0000000000000068
.text:000000000DEAD1AE cmp     rax, rbx
.text:000000000DEAD1B1 jz      short locret_DEAD1B6    ;not jmp
.text:000000000DEAD1B3 add     rsp, rdx            ;rsp = 0x000000000E0AFB20
.text:000000000DEAD261 mov     [rsi], r8        ;*rsi = input8*input8
.text:000000000DEAD212 mov     rax, [rsi]        ;rax = input8*input8
.text:000000000DEAD226 mov     rbx, [rsi]        ;rbx = input8*input8
.text:000000000DEAD145 imul    rax, rbx            ;rax = input8*input8*input8*input8
.text:000000000DEAD208 mov     [rsi], rax        ;*rsi = input8*input8*input8*input8
.text:000000000DEAD26B mov     r8, [rsi]        ;r8 = input8*input8*input8*input8

r10 = r10 /2;

.text:000000000DEAD29B mov     [rsi], r10        ;*rsi = 0x0000000000001A0C
.text:000000000DEAD212 mov     rax, [rsi]        ;rax = 0x0000000000001A0C
.text:000000000DEAD195 shr     rax, 1            ;0x0000000000000D06
.text:000000000DEAD208 mov     [rsi], rax        ;*rsi = 0x0000000000000D06
.text:000000000DEAD2A5 mov     r10, [rsi]        ;r10  = 0x0000000000000D06

if (r10 > 0)

.text:000000000DEAD1F0 pop     rbx                ;rbx = 0x0000000000000000
.text:000000000DEAD200 pop     rdx                ;rdx = 0x0000000000000068
.text:000000000DEAD1AE cmp     rax, rbx
.text:000000000DEAD1B1 jz      short locret_DEAD1B6;not jmp
.text:000000000DEAD1B3 add     rsp, rdx            ;0x000000000E0AF9F8

if (r10 & 0x01 == 0)
{
    r8 = r8*r8;
}
.text:000000000DEAD212 mov     rax, [rsi]        ;
.text:000000000DEAD1F0 pop     rbx                ;0x0000000000000001
.text:000000000DEAD177 and     rax, rbx            ;0x0000000000000000
.text:000000000DEAD208 mov     [rsi], rax        ;*rsi=0x0000000000000000
.text:000000000DEAD212 mov     rax, [rsi]        ;rax = r11 = 0x0000000000000000
.text:000000000DEAD1F0 pop     rbx                ;0x0000000000000001
.text:000000000DEAD200 pop     rdx                ;0x0000000000000068
.text:000000000DEAD1AE cmp     rax, rbx
.text:000000000DEAD1B1 jz      short locret_DEAD1B6 ; not jmp
.text:000000000DEAD1B3 add     rsp, rdx            ;0x000000000E0AFB20
.text:000000000DEAD261 mov     [rsi], r8        ;*rsi = input8*input8*input8*input8
.text:000000000DEAD212 mov     rax, [rsi]        ;rax = input8*input8*input8*input8
.text:000000000DEAD226 mov     rbx, [rsi]        ;rbx = input8*input8*input8*input8
.text:000000000DEAD145 imul    rax, rbx            ;rax =
input8*input8*input8*input8*input8*input8*input8*input8
.text:000000000DEAD208 mov     [rsi], rax
.text:000000000DEAD26B mov     r8, [rsi]        ;r8 =
input8*input8*input8*input8*input8*input8*input8*input8

r10 = r10 / 2
.text:000000000DEAD29B mov     [rsi], r10        ;0x0000000000000D06
.text:000000000DEAD212 mov     rax, [rsi]        ;rax = 0x0000000000000D06
```

```
.text:000000000DEAD195 shr     rax, 1            ;rax = 0x0000000000000683
.text:000000000DEAD208 mov     [rsi], rax        ;*rsi = 0x0000000000000683
.text:000000000DEAD2A5 mov     r10, [rsi]        ;r10 = 0x0000000000000683

if (r10 > 0)

.text:000000000DEAD212 mov     rax, [rsi]        ;rax = 0x0000000000000683
.text:000000000DEAD1F0 pop     rbx                ;0x0000000000000000
.text:000000000DEAD200 pop     rdx                ;0xFFFFFFFFFFFFFDE0
.text:000000000DEAD1AE cmp     rax, rbx
.text:000000000DEAD1B1 jz      short locret_DEAD1B6                        ;if r10/2 > 0

if (r10 & 0x01 == 1)
{
    r12 = r12 * r8;
    r8 = r8 * r8;
}

.text:000000000DEAD1B3 add     rsp, rdx            ;0x000000000E0AF9F8
.text:000000000DEAD212 mov     rax, [rsi]        ;rax = 0x0000000000000683
.text:000000000DEAD1F0 pop     rbx                ;0x0000000000000001
.text:000000000DEAD177 and     rax, rbx            ;rax = 0x0000000000000001
.text:000000000DEAD208 mov     [rsi], rax        ;*rsi = 0x0000000000000001
.text:000000000DEAD212 mov     rax, [rsi]
.text:000000000DEAD1F0 pop     rbx                ;0x0000000000000001
.text:000000000DEAD200 pop     rdx                ;0x0000000000000068
.text:000000000DEAD1AE cmp     rax, rbx
.text:000000000DEAD1B1 jz      short locret_DEAD1B6;jmped
.text:000000000DEAD1B6 retn
.text:000000000DEAD2D5 mov     [rsi], r12        ;*rsi = input8
.text:000000000DEAD212 mov     rax, [rsi]        ;rax = input8
.text:000000000DEAD261 mov     [rsi], r8        ;*rsi =
input8*input8*input8*input8*input8*input8*input8*input8
.text:000000000DEAD226 mov     rbx, [rsi]        ;rbx =
input8*input8*input8*input8*input8*input8*input8*input8
.text:000000000DEAD145 imul    rax, rbx            ;rax = input(9)
.text:000000000DEAD208 mov     [rsi], rax
.text:000000000DEAD2DF mov     r12, [rsi]        ;r12 = r12 * r8 = input(9)
.text:000000000DEAD261 mov     [rsi], r8        ;*rsi = input(8)
.text:000000000DEAD212 mov     rax, [rsi]
.text:000000000DEAD226 mov     rbx, [rsi]
.text:000000000DEAD145 imul    rax, rbx
.text:000000000DEAD208 mov     [rsi], rax
.text:000000000DEAD26B mov     r8, [rsi]


r10 = r10 / 2;

.text:000000000DEAD29B mov     [rsi], r10
.text:000000000DEAD212 mov     rax, [rsi]
.text:000000000DEAD195 shr     rax, 1
.text:000000000DEAD208 mov     [rsi], rax
.text:000000000DEAD2A5 mov     r10, [rsi]
.text:000000000DEAD29B mov     [rsi], r10
.text:000000000DEAD212 mov     rax, [rsi]
.text:000000000DEAD1F0 pop     rbx                ; 0x0000000000000000
```

将我们写的伪码汇总一下，大概如下：

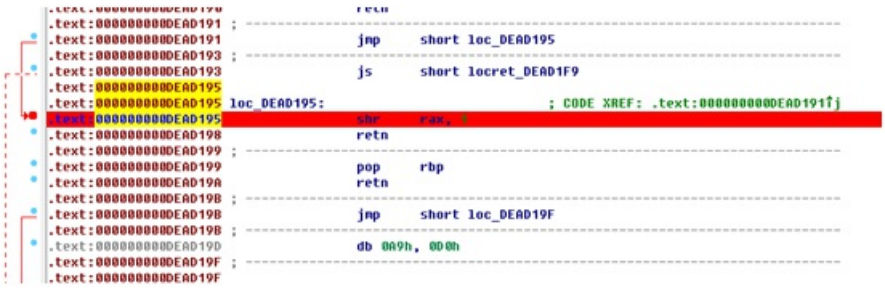
```
r9 = 0x1337DEADBEEF0095;

r9 = 0x000000000000CAFE + 0x000000000000BEEF

r12 = 1, r10 = 13337

if (r10 & 0x01 == 0x01)
{
    r12 = r12 * r8;
    r8 = r8 * r8;
}

r10 = r10 / 2;

if (r10 != 0)

if (r10 & 0x01 == 0)
{
    r8 = r8 * r8;
}

r10 = r10 /2;

if (r10 != 0)

if (r10 & 0x01 == 0)
{
    r8 = r8*r8;
}

r10 = r10 / 2

if (r10 ! 0)

if (r10 & 0x01 == 1)
{
    r12 = r12 * r8;
    r8 = r8 * r8;
}

r10 = r10 / 2;
```

从上述伪码大概可以看，开始的时候，程序进行了初始化的赋值操作，然后会对r10 & 0x01进行判断，如果该值不为0，会执行“r12 = r12 * r8; r8 = r8 * r8;”，如果该值为0，执行“r8 = r8 * r8;”，然后r10右移一位，重复这个过程，直到r10为0为止。

那么用C语言表示这段代码的逻辑，如下：

```
r9 = 0x1337DEADBEEF0095;

r12 = 1
r9 = r9 * 0x000000000000CAFE + 0x000000000000BEEF;
for (r10 = 13337; r10 != 0; r10 = r10/2)
{
    if (r10 & 0x01)
    {
            r12 = r12 * r8
    }

    r8 = r8 * r8;
}
```

上述处理都是r10不为0的时候。那么当r10为0的时候，处理逻辑是怎样的呢？由于r10的初始值为13337，经过13次右移1位后，r10将为0，我们在移位处设置断点：



重新运行程序进行调试，前13次断在该位置时，我们直接按”F9”，在第14次断在该位置时，我们按”F8”进行单步跟踪，此时有效执行的gadgets及分析如下：

```
if (r10 == 0 )
.text:000000000DEAD208 mov     [rsi], rax        ;*rsi = 0
.text:000000000DEAD2A5 mov     r10, [rsi]        ;r10 = 0
.text:000000000DEAD212 mov     rax, [rsi]        ;rax = 0
.text:000000000DEAD1F0 pop     rbx                ;0x0000000000000000
.text:000000000DEAD200 pop     rdx                ;0xFFFFFFFFFFFFFDE0
.text:000000000DEAD1AE cmp     rax, rbx            ;
.text:000000000DEAD1B1 jz      short locret_DEAD1B6

if (r12 != r9)
{
    handlemsg ()
}

.text:000000000DEAD2D5 mov     [rsi], r12
.text:000000000DEAD212 mov     rax, [rsi]
.text:000000000DEAD27E mov     [rsi], r9
.text:000000000DEAD226 mov     rbx, [rsi]
.text:000000000DEAD200 pop     rdx
.text:000000000DEAD1AE cmp     rax, rbx
.text:000000000DEAD1B1 jz      short locret_DEAD1B6
.text:000000000DEAD200 pop     rdx                        ;0xFFFFFFFFFFFFFC38
.text:000000000DEAD1DF loop    loc_DEAD1DB
```

这段用伪码来表示的执行逻辑为：

```
if (r10 == 0)
{
    if (r12 != r9)
    {
        handlemsg ()
    }
}
```

即在上述for循环结束时，判断r12是否等于r9，如果不相等，则继续监听socket，重复整个过程。

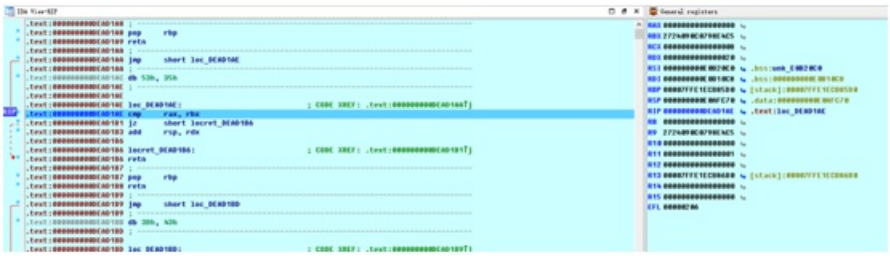
至此，用c语言还原的完整代码为：

```
r9 = 0x1337DEADBEEF0095;

r12 = 1
r9 = r9 * 0x000000000000CAFE + 0x000000000000BEEF;
for (r10 = 13337; r10 != 0; r10 = r10/2)
{
    if (r10 & 0x01)
    {
            r12 = r12 * r8
    }

    r8 = r8 * r8;
}

if (r9 != r12)
{
    handlemsg()
}
```

那么还有一个分支我们没有走到，即r9如果等于r12，程序会执行什么操作？还记得之前POP RCX后，RCX==8吗？我们猜想此时会进入外层的这个循环。动态调试一下来验证我们的想法。重复上述过程，在第14次断在.text:000000000DEAD195 shr　rax, 1时，采用单步调试，在执行到下面的判断时，我们可以修改寄存器的值来满足这个判断，进而执行r9==r12后的逻辑：

```
.text:000000000DEAD1AE cmp     rax, rbx
.text:000000000DEAD1B1 jz      short locret_DEAD1B6
```

在执行到这个位置后，寄存器内容如下：



此时rax为0，rbx为0x2724090C0798E4C5，我们修改rax寄存器的值为0x2724090C0798E4C5，这样就满足了r9==r12，修改后，我们跟踪gadgets的执行，有效指令如下：

```
text:000000000DEAD1FC jmp     short loc_DEAD200
.text:000000000DEAD200 pop     rdx
.text:000000000DEAD1DF loop    loc_DEAD1DB
.text:000000000DEAD1DB add     rsp, rdx
.text:000000000DEAD127 mov     rax, [rdi]
.text:000000000DEAD208 mov     [rsi], rax
.text:000000000DEAD26B mov     r8, [rsi]
```

此时，rcx变为7，rdi指向用户输入的下一个8字节，开始了一个新的循环过程。

在8个外循环执行后，程序将打印出flag。

综合上述分析，我们可以用C语言表示该程序的执行逻辑了：

```
#include <stdio.h>
#include <stdlib.h>

long long r8[8] = {???, ???, ???, ???, ???, ???, ???, ???};

int main ()
{
    r9 = 0x1337DEADBEEF0095;

    for (rcx = 0; rcx < 8; rcx ++)
    {
        r9 = r9 * 0x000000000000CAFE + 0x000000000000BEEF;
        r12 = 1;

        for (r10 = 13337; r10 > 0; r10 = r10 / 2)
        {
            if (r10 & 0x01)
                r12 = r12 * r8[rcx];

            r8[rcx] = r8[rcx] * r8[rcx];
        }

        if (r12 != r9)
        {
            handlemsg();
        }
    }

    printflag ();
}
```

R9的初始值为0x1337DEADBEEF0095，每次循环后会变化，我们可以计算出每次循环后R9的值分别为：

```
long long r9[8] = {
    0x2724090c0798e4c5,
    0x44e477ee2e372c65,
    0xa150eec963c67d25,
    0xeab7d48b9db01ba5,
    0xf01b0cf36a8c5ea5,
    0x930eeb9679f4d8a5,
    0xaeb27b8833e1e4a5,
    0x2a900a13b88bcca5
};
```

从我们分析的结果来看，R12为用户输入数据的N次幂，N是多少呢？仿照这个程序我们写一段C代码来计算一下：

```
int getpower ()
{
    long long magic = 0x0000000000003419;
    int twopower = 1;
    int r8power = 0;

    for (; magic > 0; magic = magic >> 1)
    {
        if (magic & 0x01)
        {
            r8power += twopower;
        }

        twopower = twopower*2;
    }

    return r8power;
}
```

可以知道幂的次数是13337，也就是程序监听的端口。于是问题转化为，给定Y，求X，使得X满足如下条件：

```
X^13337 = Y
```

实际上，由于X是一个64位整数，在计算其幂时，很可能会溢出，所以实际问题应该是给定Y，求X，使得X满足如下条件：

```
X^13337 %2^64 = Y
```

这就是个数学问题了，本人数学功底太差，不知道怎么去求解。但是看网上有人用欧拉定理来求解，有人用广义费马小定理来求解，膜拜这些数学大牛们。

网上还看到有大牛们用python的pack函数来求解，得到了flag，没想明白是怎么做到的，代码如下：

```python
import sys,struct

def f(x):
    return struct.pack('Q', x)

p = ''
p += f(0xd5b028b6c97155a5)
p += f(0x51a2c3e8e288fa45)
p += f(0x561720a3f926b105)
p += f(0xa325ec548e4e0385)
p += f(0x5369761ad6ccde85)
p += f(0x9475802813002885)
p += f(0xcadd6a0bdc679485)
p += f(0x7d67b37124bcbc85)

sys.stdout.write(p)
```

将上述脚本的输出传递给r0ops即可打印flag:

```
root@kali64:/home/ctf/0ctf2015/r0ops# python roops.py | nc localhost 13337

root@kali64:/home/ctf/0ctf2015/r0ops# ./r0ops
YOU WIN!

FLAG IS: 0ctf{c97155a5e288fa45f926b1058e4e0385d6ccde8513002885dc67948524bcbc85}
```

转载于:https://www.cnblogs.com/gsharpsh00ter/p/6441433.html