2. 冷补丁 getshell

2.1 实验要求

- 程序getshell调用printf()函数打印了/bin/sh字符串，利用这一点可以实现getshell。
- 要求利用LIEF库，将getshell程序中的printf() 函数替换为补丁函数newprintf()，获取系统的控制权，在当前目录下写入包含学号的文件，再将该文件显示出来。

2.2 实验过程

2.2.1 程序分析

- 首先使用IDA打开getshell程序，其main函数的反汇编结果如下：

```
; Attributes: bp-based frame

; int __cdecl main(int argc, const char **argv, const char **envp)
public main
main proc near
; __unwind {
push    rbp
mov     rbp, rsp
lea     rax, format     ; "/bin/sh"
mov     rdi, rax        ; format
mov     eax, 0
call    _printf
lea     rax, s          ; "\ngetshell"
mov     rdi, rax        ; s
call    _puts
mov     eax, 0
pop     rbp
retn
; } // starts at 401136
main endp
```

- 将main函数反编译，结果如下：

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  printf("/bin/sh", argv, envp);
  puts("\ngetshell");
  return 0;
}
```

2.2.2 补丁编写

- 由于要将`printf()`替换为补丁函数 `newprintf()`以实现`getshell`，且已有`/bin/sh`字符串可作为`getshell`时的可用参数（实际上，如果没有该字符串，也可以使用汇编语句在栈中手动构造该字符串），因此考虑在`newprintf()`中使用`sys_execve`系统调用，`x86_64`架构下`sys_execve`系统调用的相关信息如下：

rax	System Call	rdi	rsi	rdx
59	sys_execve	const char* filename	const char* argv[]	const char* envp[]

- 据此构造`newprintf()`如下：

```c
// hook_printf.c
void newprintf() {
    asm (
        "movq $0,  %rsi\n"
        "movq $0,  %rdx\n"
        "movq $59, %rax\n"
        "syscall\n"
    );
}
```

- 编译该补丁文件生成一个静态函数库（可使用`gcc -S hook_printf.c`查看编译得到的汇编指令）：

```
gcc -m64 -fPIC --shared hook_printf.c -o hook_printf
```

2.2.3 程序修改

- 安装`python`的`lief`和`pwn`库：

```
pip3 install lief
pip3 install pwn
```

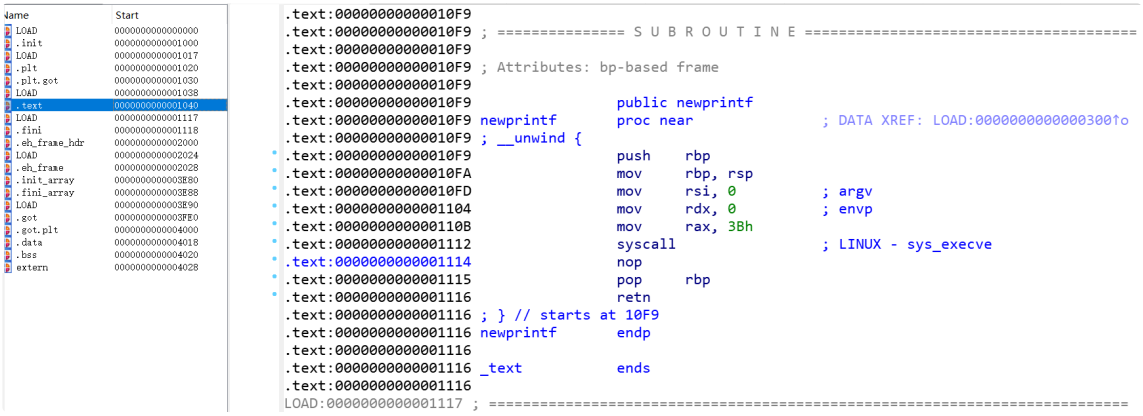
- 修改脚本文件`getshell.py`如下：

```python
#!/usr/bin/python3
from pwn import *
import lief

# Patch a call instruction at srcaddr whose target is dstaddr.
def patch_call(file, srcaddr, dstaddr, arch='amd64'):
```

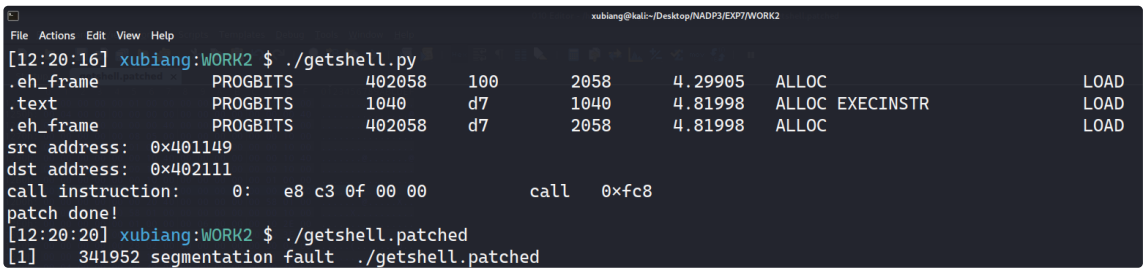
```python
    print('src address: ', hex(srcaddr), '\ndst address: ', hex(dstaddr))
    relative_offset = p32((dstaddr - (srcaddr + 5)) & 0xffffffff)
    call_inst = '\xe8'.encode('latin1') + relative_offset
    print('call instruction: ', disasm(call_inst, arch=arch))
    file.patch_address(srcaddr, [i for i in call_inst])

binary = lief.parse('./getshell')
hook = lief.parse('./hook_printf')

sec_ehframe = binary.get_section('.eh_frame')
print(sec_ehframe)
sec_text = hook.get_section('.text')
print(sec_text)
sec_ehframe.content = sec_text.content
print(binary.get_section('.eh_frame'))

srcaddr = 0x401149
dstaddr = sec_ehframe.virtual_address + 0xb9

patch_call(binary, srcaddr, dstaddr)

binary.write('getshell.patched')
print('patch done!')
```

- 以上脚本中srcaddr为getshell中call _printf指令的地址，dstaddr为hook_printf中newprintf()函数的起始地址，即.text段的起始地址加偏移量0xb9，使用patch_call()函数构造srcaddr处向newprintf()的一条call指令：

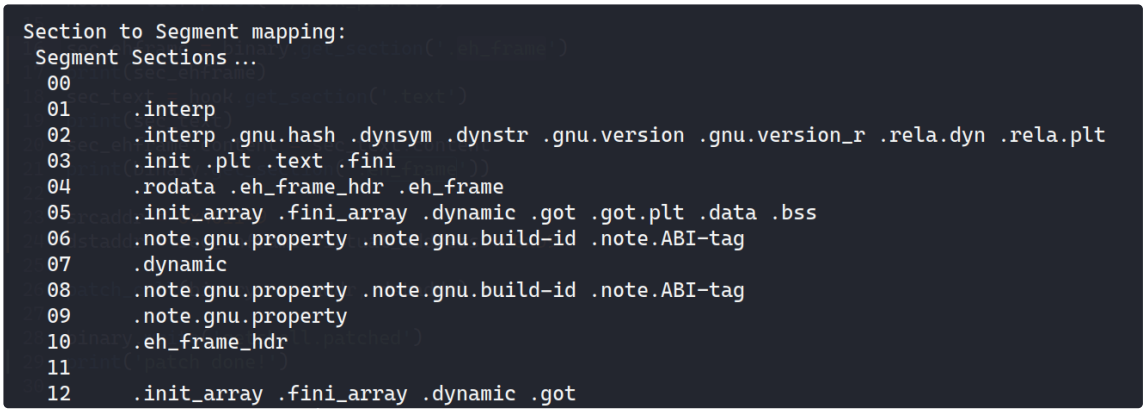
```
.text:0000000000401136 ; =============== S U B R O U T I N E =======================================
.text:0000000000401136
.text:0000000000401136 ; Attributes: bp-based frame
.text:0000000000401136
.text:0000000000401136 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:0000000000401136                 public main
.text:0000000000401136 main            proc near               ; DATA XREF: _start+1D↑o
.text:0000000000401136 ; __unwind {
.text:0000000000401136                 push    rbp
.text:0000000000401137                 mov     rbp, rsp
.text:000000000040113A                 lea     rax, format     ; "/bin/sh"
.text:0000000000401141                 mov     rdi, rax        ; format
.text:0000000000401144                 mov     eax, 0
.text:0000000000401149                 call    _printf
.text:000000000040114E                 lea     rax, s          ; "\ngetshell"
.text:0000000000401155                 mov     rdi, rax        ; s
.text:0000000000401158                 call    _puts
.text:000000000040115D                 mov     eax, 0
.text:0000000000401162                 pop     rbp
.text:0000000000401163                 retn
.text:0000000000401163 ; } // starts at 401136
.text:0000000000401163 main            endp
.text:0000000000401163
.text:0000000000401163 ; ---------------------------------------------------------------------------
```



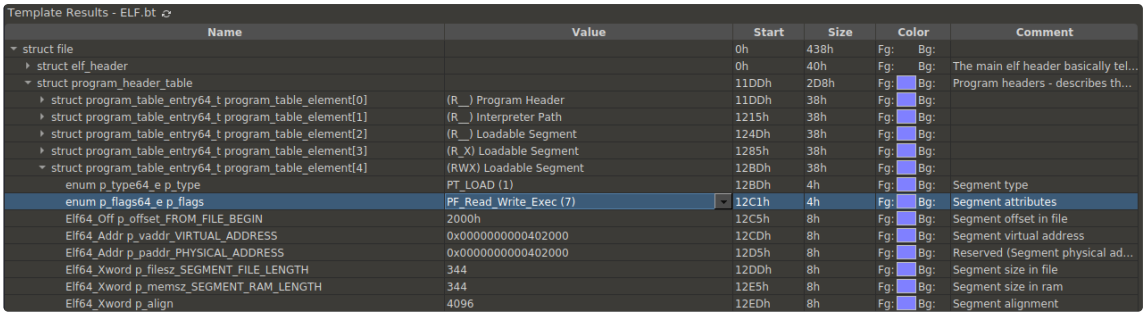
- 运行以上脚本并生成 `getshell.patched` 后，尝试运行 `getshell.patched`，但由于 `.eh_frame` 段的权限为只读导致段错误：



- 为了解决段错误，首先使用 `readelf --segments getshell.patched` 指令查看 `getshell.patched` 的段信息，可知 `Section .eh_frame` 所在的段为 `Segment 04`：

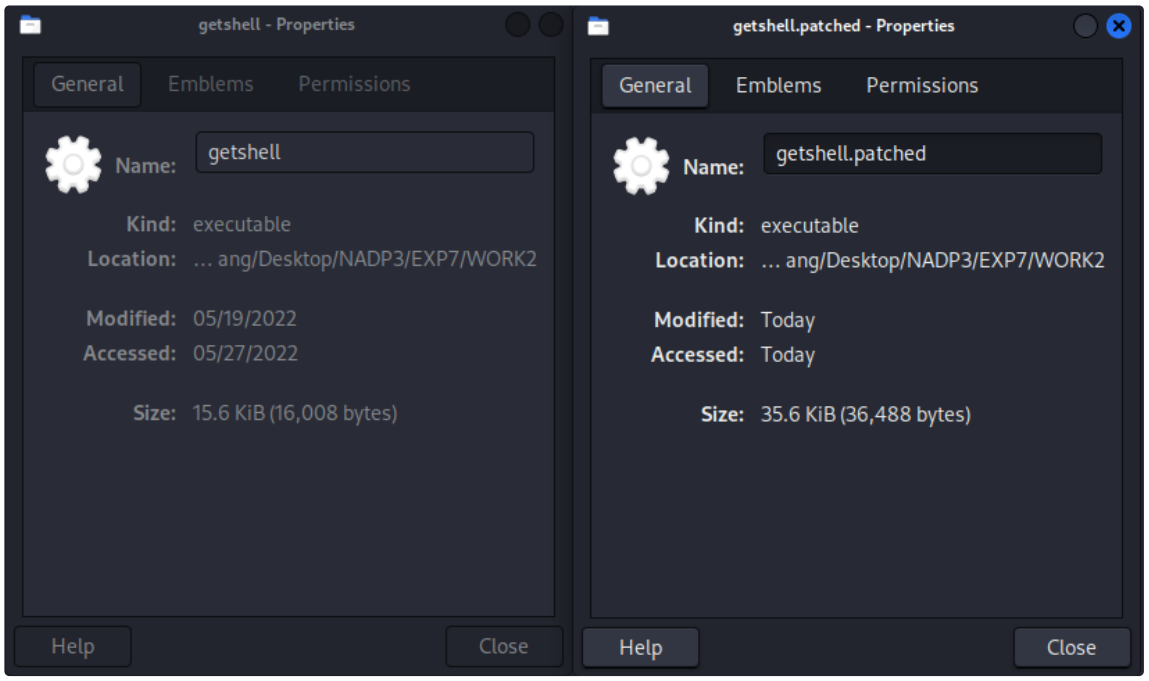


- 安装 `010 Editor`，打开 `getshell.patched` 并导入模板 `ELF.bt` 对该 `ELF` 文件进行解析，将 `Segment 04` 对应的权限设为读写执行并保存：



- 再次尝试运行 `getshell.patched`，可以正常获得 `shell` 并完成相关操作（获得的 `shell` 的权限与执行程序的用户权限、程序的 `setuid` 位、`/bin/sh` 链接的 `shell` 程序等因素有关，不再深入探究，此处使用 `root` 权限运行程序以获得具有 `root` 权限的 `shell`）：

- 使用这种方式得到的程序大小有变化，原程序大小为 `16008 Bytes`，修改后的程序大小为 `36,488 Bytes`，这是 `LIEF` 库导致的：



2.2.4 手动插入补丁代码

- 将以上生成的 `newprintf()` 函数中的汇编代码手动插入到程序的 `.eh_frame` 段即可在不改变程序大小的前提下完成补丁，使用以上使用 `IDA` 查看的 `getshell` 的结果可知 `call _printf` 指令的地址为 `0x401149`，吓一跳指令的地址为 `0x40114E`，需要插入的汇编代码如下（省略了函数调用的相关指令）：

```
; rdi 在原程序中已为 /bin/sh 的地址
mov rsi, 0        ; rsi ← NULL
mov rdx, 0        ; rdi ← NULL
mov rax, 59       ; rax ← 59 (sys_execve)
syscall
jmp 0x40114E      ; jmp back
```

- 在 `IDA` 中查看原程序的段表，并查看 `.eh_frame` 段，选择 `0x402088` 处作为补丁代码的起始位置：

Name	Start
LOAD	0000000000400000
.init	0000000000401000
LOAD	0000000000401017
.plt	0000000000401020
.text	0000000000401050
LOAD	00000000004011D1
.fini	00000000004011D4
.rodata	0000000000402000
LOAD	0000000000402016
.eh_frame_hdr	0000000000402018
LOAD	0000000000402054
.eh_frame	0000000000402058
.init_array	0000000000403E10
.fini_array	0000000000403E18
LOAD	0000000000403E20
.got	0000000000403FF0
.got.plt	0000000000404000
.data	0000000000404028
.bss	0000000000404038
.prgend	0000000000404040
extern	0000000000404048

```
.eh_frame:0000000000402086                 db    0
.eh_frame:0000000000402087                 db    0
.eh_frame:0000000000402088                 db    0
.eh_frame:0000000000402089                 db    0
.eh_frame:000000000040208A                 db    0
.eh_frame:000000000040208B                 db    0
.eh_frame:000000000040208C                 db    1
.eh_frame:000000000040208D                 db  7Ah ; z
.eh_frame:000000000040208E                 db  52h ; R
.eh_frame:000000000040208F                 db    0
.eh_frame:0000000000402090                 db    1
.eh_frame:0000000000402091                 db  78h ; x
.eh_frame:0000000000402092                 db  10h
.eh_frame:0000000000402093                 db    1
.eh_frame:0000000000402094                 db  1Bh
.eh_frame:0000000000402095                 db  0Ch
.eh_frame:0000000000402096                 db    7
```

- 将补丁的汇编代码使用 `Keypatch` 的 `Patcher` 或 `Fill range` 功能填充到原程序 `.eh_frame` 段的相应位置，点击 `C` 将其转换为代码：

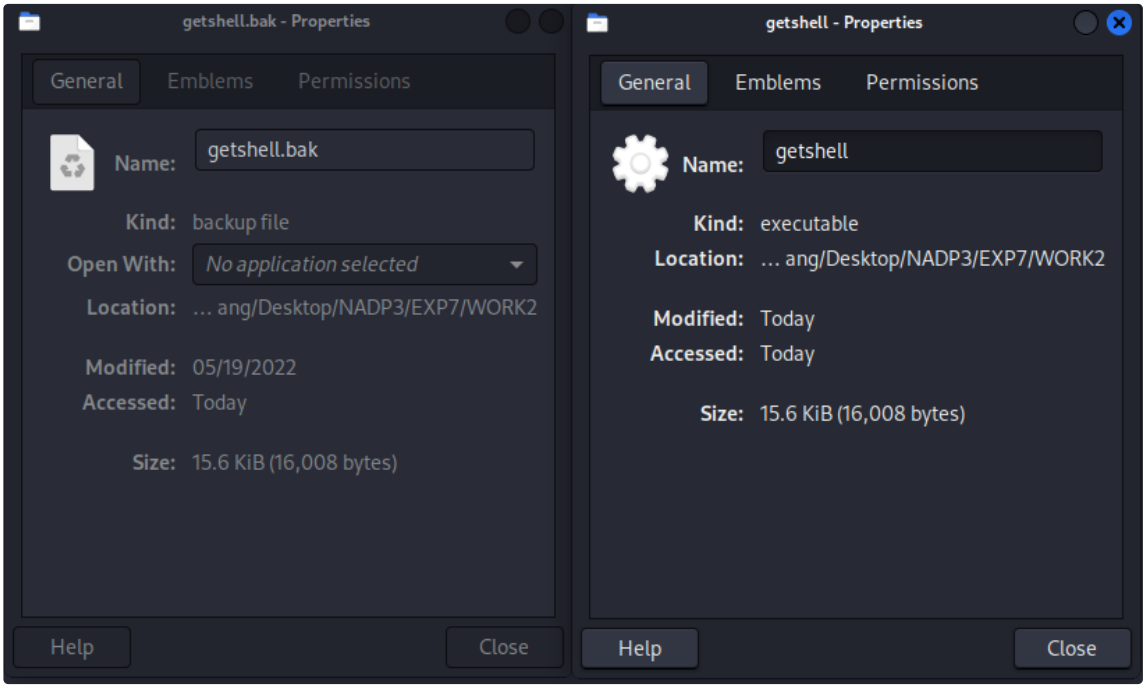
```
.eh_frame:0000000000402088 ; ---------------------------------------------------------------------------
.eh_frame:0000000000402088                 mov     rsi, 0          ; Keypatch modified this from:
.eh_frame:0000000000402088                                         ;   db 0
.eh_frame:0000000000402088                                         ;   db 0
.eh_frame:0000000000402088                                         ;   db 0
.eh_frame:0000000000402088                                         ;   db 0
.eh_frame:0000000000402088                                         ;   db 1
.eh_frame:0000000000402088                                         ;   db 7Ah
.eh_frame:0000000000402088                                         ;   db 52h
.eh_frame:000000000040208F                 mov     rdx, 0          ; Keypatch modified this from:
.eh_frame:000000000040208F                                         ;   db 0
.eh_frame:000000000040208F                                         ;   db 1
.eh_frame:000000000040208F                                         ;   db 78h
.eh_frame:000000000040208F                                         ;   db 10h
.eh_frame:000000000040208F                                         ;   db 1
.eh_frame:000000000040208F                                         ;   db 1Bh
.eh_frame:000000000040208F                                         ;   db 0Ch
.eh_frame:0000000000402096                 mov     rax, 3Bh        ; Keypatch modified this from:
.eh_frame:0000000000402096                                         ;   db 7
.eh_frame:0000000000402096                                         ;   db 8
.eh_frame:0000000000402096                                         ;   db 90h
.eh_frame:0000000000402096                                         ;   db 1
.eh_frame:0000000000402096                                         ;   db 0
.eh_frame:0000000000402096                                         ;   db 0
.eh_frame:0000000000402096                                         ;   db 10h
.eh_frame:000000000040209D                 syscall                 ; Keypatch modified this from:
.eh_frame:000000000040209D                                         ;   db 0
.eh_frame:000000000040209D                                         ;   db 0
.eh_frame:000000000040209F                 jmp     loc_40114E      ; Keypatch modified this from:
.eh_frame:000000000040209F                                         ;   db 0
.eh_frame:000000000040209F                                         ;   db 1Ch
.eh_frame:000000000040209F                                         ;   db 0
.eh_frame:000000000040209F                                         ;   db 0
.eh_frame:000000000040209F                                         ;   db 0
.eh_frame:000000000040209F ; ---------------------------------------------------------------------------
```

- 再将源程序的 `call _printf` 指令使用 `Keypatch` 修改为 `jmp 0x402088` 即可：

```
.text:0000000000401136 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:0000000000401136                 public main
.text:0000000000401136 main            proc near               ; DATA XREF: _start+1D↑o
.text:0000000000401136
.text:0000000000401136 ; FUNCTION CHUNK AT .eh_frame:0000000000402088 SIZE 0000001C BYTES
.text:0000000000401136
.text:0000000000401136 ; __unwind {
.text:0000000000401136                 push    rbp
.text:0000000000401137                 mov     rbp, rsp
.text:000000000040113A                 lea     rax, filename   ; "/bin/sh"
.text:0000000000401141                 mov     rdi, rax        ; filename
.text:0000000000401144                 mov     eax, 0
.text:0000000000401149                 jmp     loc_402088      ; Keypatch modified this from:
.text:0000000000401149                                         ;   call _printf
.text:000000000040114E ; ---------------------------------------------------------------------------
.text:000000000040114E
.text:000000000040114E loc_40114E:                             ; CODE XREF: main+F69↓j
.text:000000000040114E                 lea     rax, s          ; "\ngetshell"
.text:0000000000401155                 mov     rdi, rax        ; s
.text:0000000000401158                 call    _puts
.text:000000000040115D                 mov     eax, 0
.text:0000000000401162                 pop     rbp
.text:0000000000401163                 retn
.text:0000000000401163 ; } // starts at 401136
.text:0000000000401163 main            endp
```

- 将以上修改应用到源文件并使用上述修改段属性的方法修改 `.eh_frame` 段的权限后进行测试，可以正常完成功能：

- 查看该修改后的程序 `getshell` 与原程序 `getshell.bak` 的大小均为 `16008 Bytes`，大小没有发生变化：



2.2.5 使用Preload Hook

- 使用 `Preload Hook` 可以不对程序本身做任何修改，因此也不会改变程序大小，编写自定义的 `printf()` 函数如下：

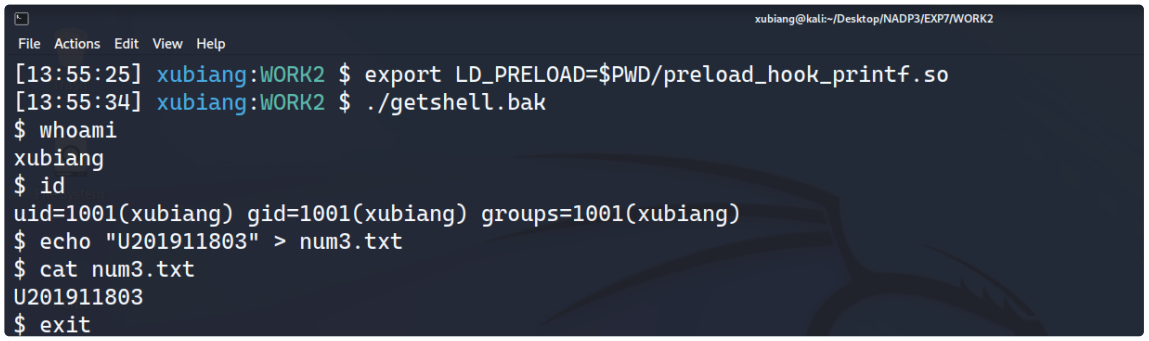
```c
// preload_hook_printf.c
#define _GNU_SOURCE
#include <sys/stat.h>
#include <unistd.h>
#include <dlfcn.h>
#include <string.h>
#include <stdlib.h>

int printf(char *a, int b) {
    if (!strcmp(a, "/bin/sh")) {
        system(a);
```

```c
    }
    int (*old_printf)(char *, int);
    old_printf = (int (*)(char *, int))dlsym(RTLD_NEXT, "printf");
    old_printf(a, b);
    puts("\n");
}
```

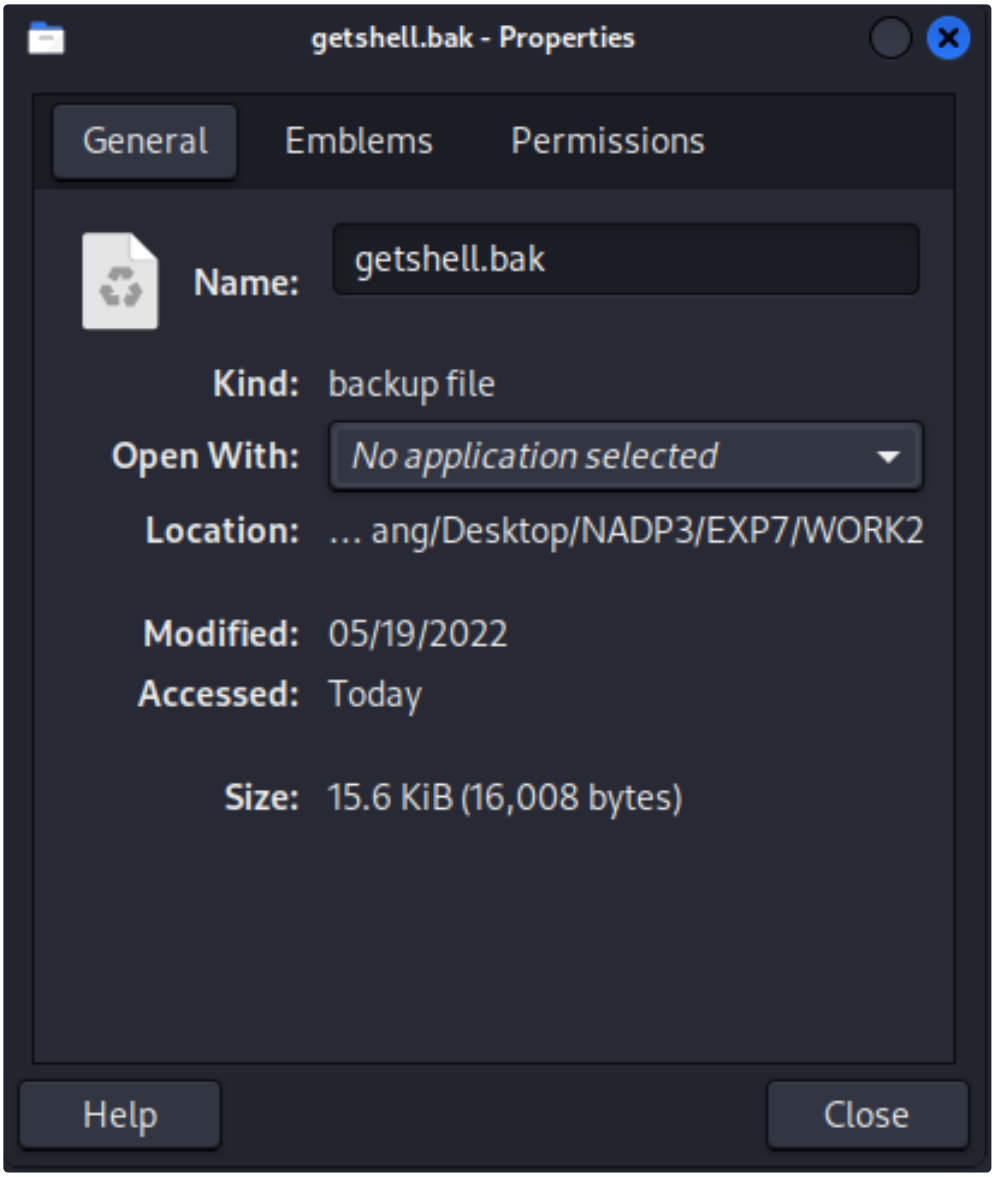
- 编译该文件生成动态链接库：

```
gcc -m64 -fPIC --shared preload_hook_printf.c -o preload_hook_printf.so -ldl
```

- 加载动态链接库`export LD_PRELOAD=$PWD/preload_hook_printf.so`，并执行`getshell.bak`程序，可以正常完成功能：



- 由于未对`getshell.bak`程序做任何更改，其大小未发生改变，仍为`16008 Bytes`：


getshell.bak - Properties
General
Emblems
Permissions
Name:
getshell.bak
Kind:
backup file
Open With:
No application selected
Location:
... ang/Desktop/NADP3/EXP7/WORK2
Modified:
05/19/2022
Accessed:
Today
Size:
15.6 KiB (16,008 bytes)
Help
Close