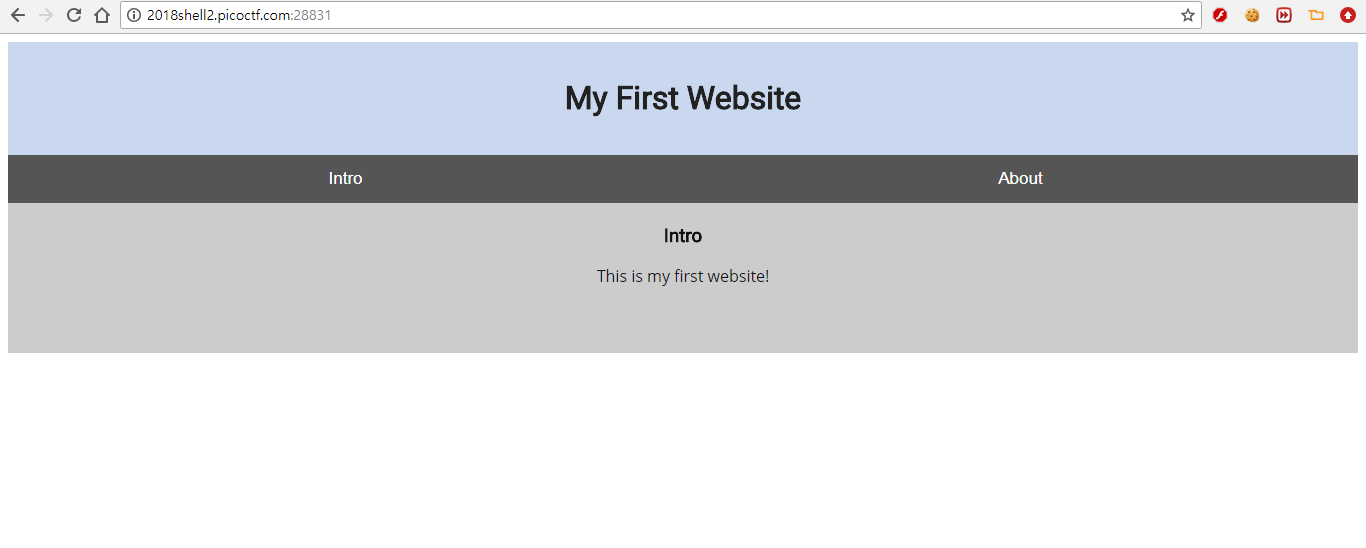
Write-Up

Pico CTF

양은석

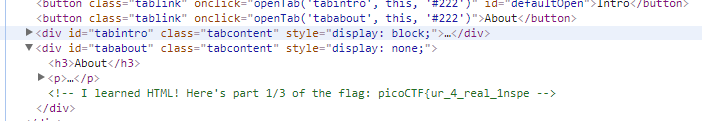
1. Inspect Me



문제의 링크를 들어가면 위와 같은 페이지가 보이고, 버튼들에는 아무 기능이 없다.

개발자 옵션을 키고 소스코드를 확인해보면





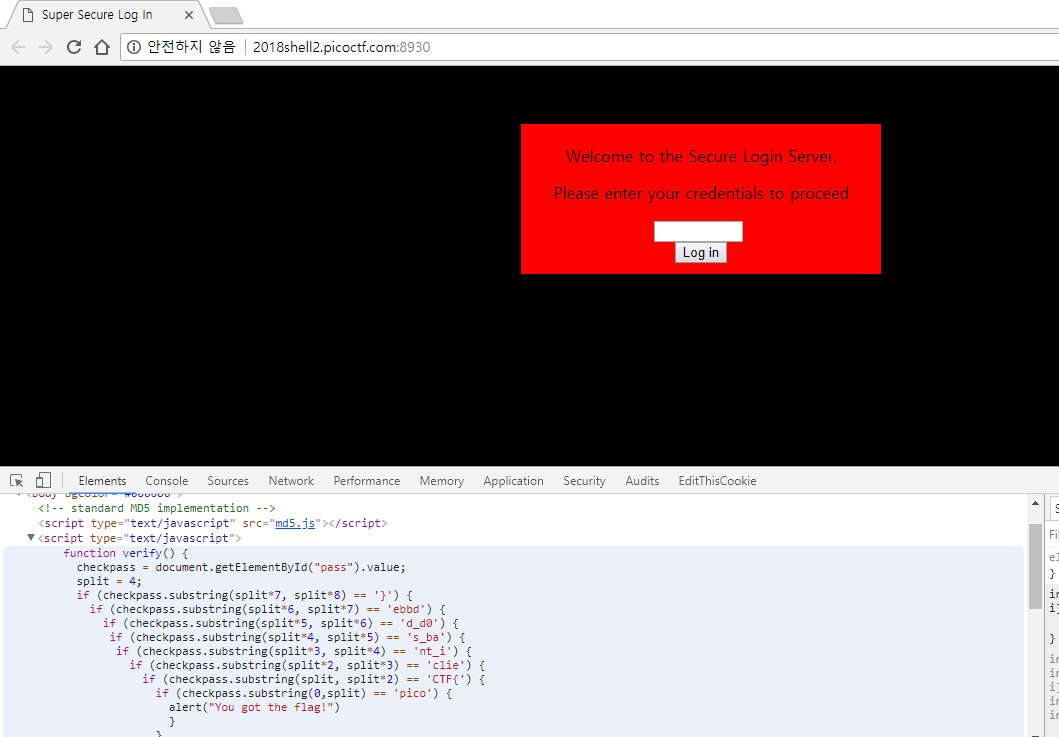
위와 같이 css, js문서가 연결되어 있고, 아래에는 주석으로 플래그의 일부분이 적혀있다. Css와 js 문서를 마저 확인해보면

웹4.PNG

Css에서 플래그의 나머지 부분을 얻을 수 있다.

Flag : picoCTF{ur\_4\_real\_1nspect0r\_g4dget\_b4887011}

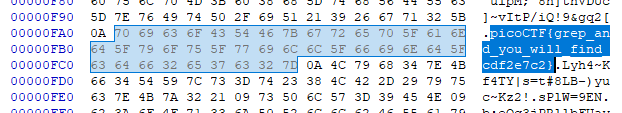
1. Client Side is still bad



링크를 확인하면 위와 같이 입력창이 하나 나오는데 위의 문제와 마찬가지로 소스코드를 확인하면 평문으로 플래그가 적혀있는 모습을 볼 수 있다.

Flag : picoCTF{client\_is\_bad\_d0ebbd}

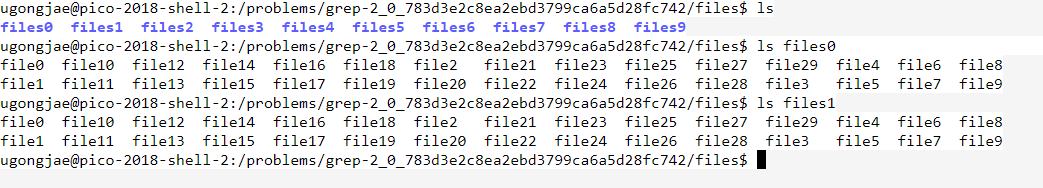
1. grep

문제에 왼쪽과 같은 파일 하나가 첨부되어 있다. 이 파일의 hex값을 뷰어로 확인해보면

위와 같이 중간에 플래그값이 들어있는 것을 알 수 있다.

Flag : picoCTF{grep\_and\_you\_will\_find\_cdf2e7ce}

1. grep2



쉘을 통해 해당 문제의 디렉터리로 들어가면 여러 디렉터리로 나뉘어져 있고 각각의 디렉터리 아래에 여러 파일들이 있는 모습을 볼 수 있다.



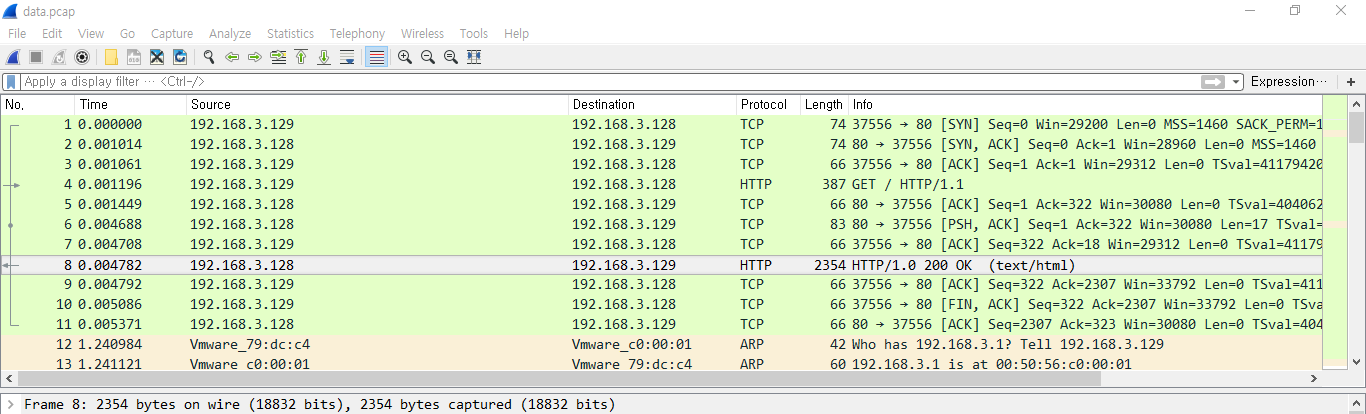
각각의 디렉터리에 대해 플래그의 앞부분인 picoCTF를 이용해

Strings \* | grep pico

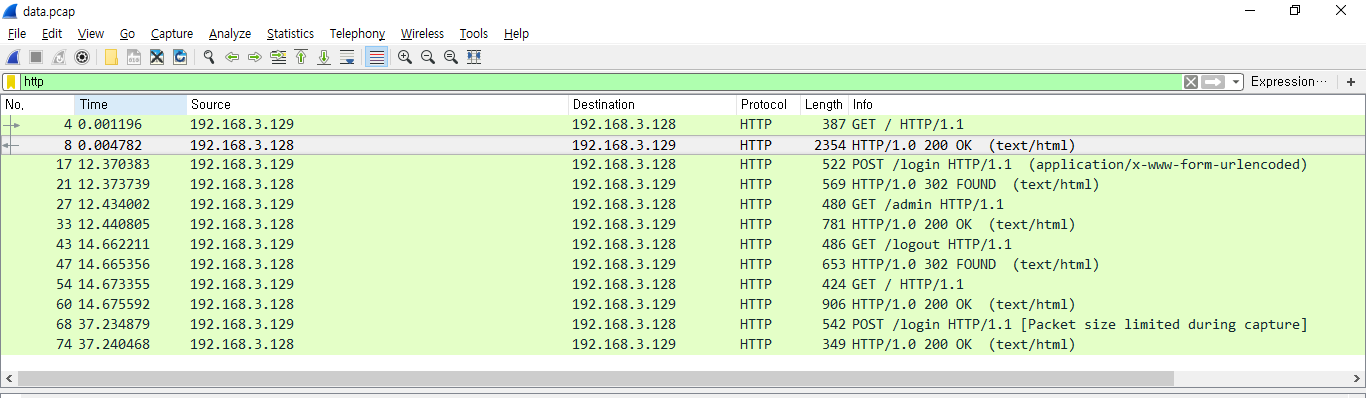
으로 추출하면 플래그를 확인할 수 있다.

Flag : picoCTF{grep\_r\_and\_you\_will\_find\_24c911ab}

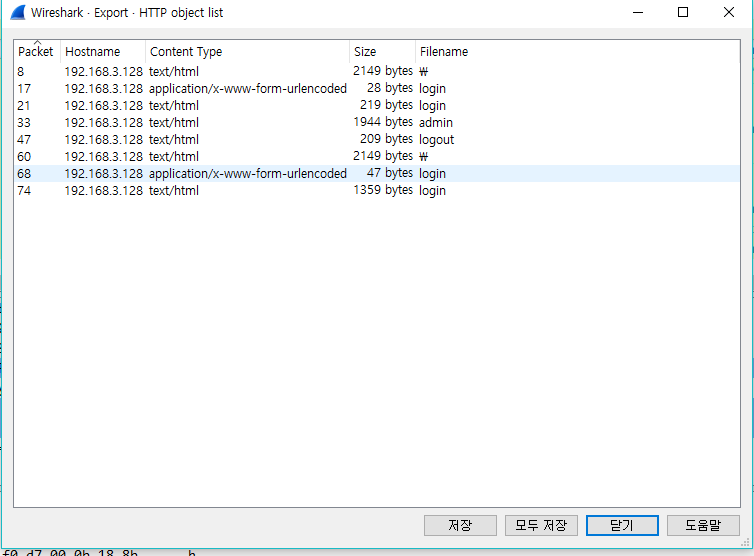
1. admin\_panel



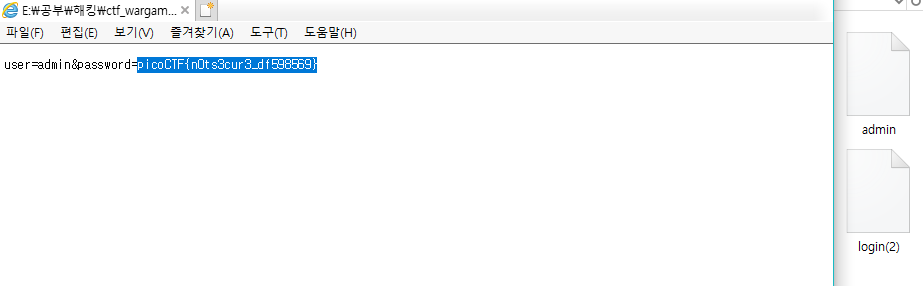
문제에서 다음과 같은 pcap 파일이 주어진다.



필터링을 통해 http패킷만 확인을 해보면 여러 로그인 기록이 보인다.



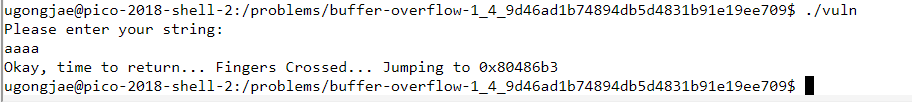
위와 같이 http패킷들을 추출한다.



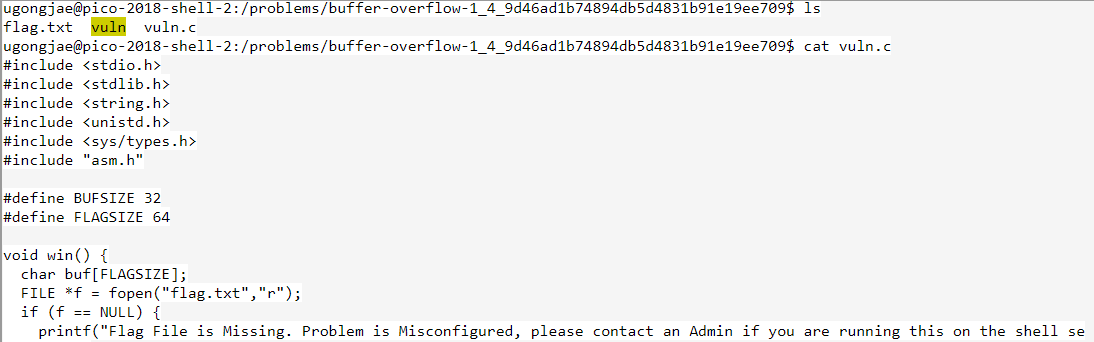
그러면 여러 패킷들 중 login(2)라는 패킷에서 비밀번호에 플래그가 들어가있는 것을 확인할 수 있다.

Flag : picoCTF{n0ts3cur3\_df598569}

1. buffer overflow 1

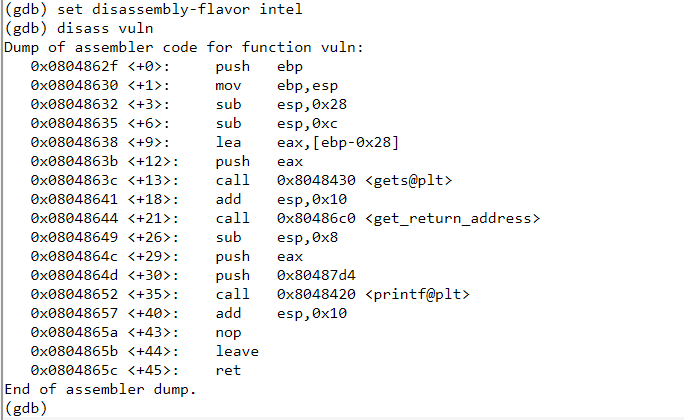


쉘을 통해 해당 디렉터리에서 vuln이란 이름의 실행파일을 확인할 수 있다.

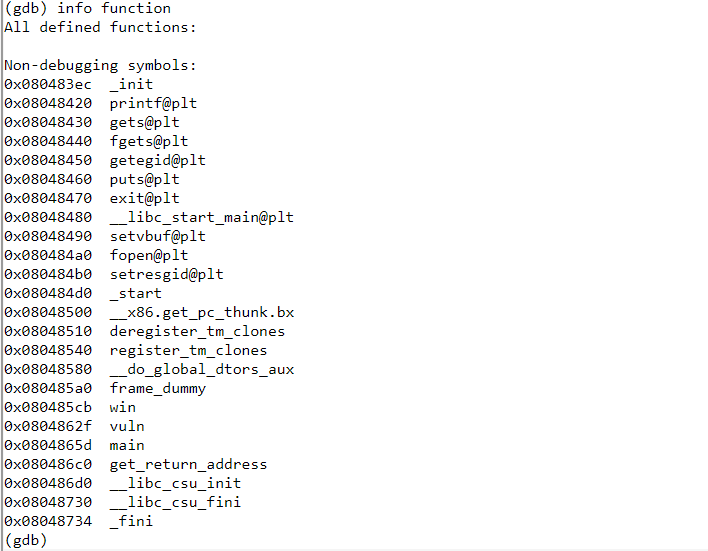




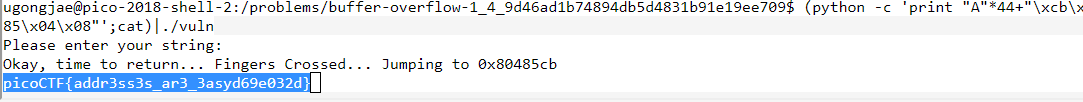
소스코드를 확인하면 vuln()함수 안에 gets라는 취약함수가 보인다. 이를 통해 버퍼를 오버플로우 시켜 win()함수를 불러오면 플래그를 얻을 수 있을 것으로 보인다.



Gdb를 통해 버퍼의 크기를 우선 확인한다. 0x08048638<+9>부분에서 ebp-0x28부터 버퍼가 시작됨을 볼 수 있다. 즉 40바이트의 버퍼를 할당 받았으므로 4바이트의 sfp부분을 더해서 44바이트를 채우고 나면 스택의 ret부분을 오염시킬 수 있을 것이다.



Info function 명령어를 통해 함수들의 시작주소를 확인한다 위에서 ret를 오염시킬 때 win의 시작주소인 0x080485cb 로 오염시킨다.

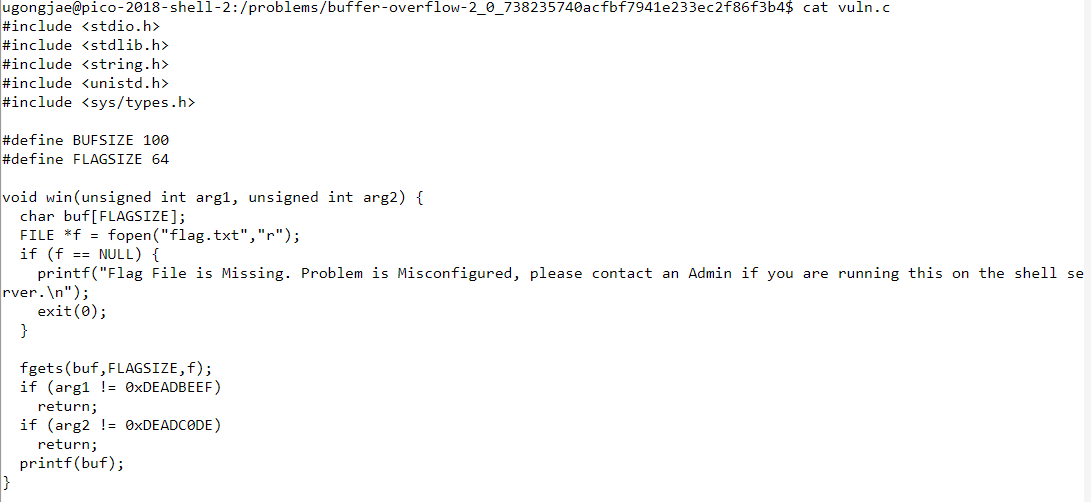


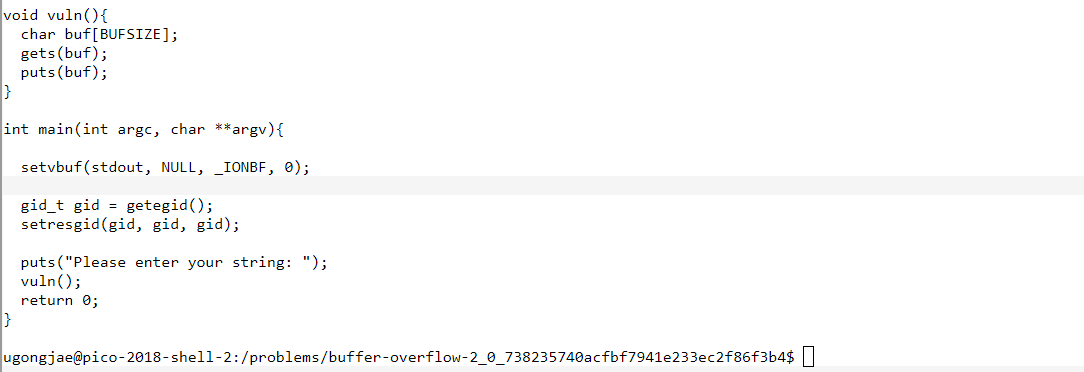
(python –c ‘print “A”\*44+\xcb\x85\x04\x08”’cat)|./vuln

위와 같은 코드로 취약점을 공격할 수 있다.

Flag : picoCTF{addr3ss3s\_ar3\_3asyd69e032d}

1. buffer overflow 2





이전 문제와 비슷하게 입력을 받고 출력을 하는 프로그램이다. 마찬가지로 vuln()함수에 있는 gets() 함수를 공략할 것이다.

위의 문제처럼 버퍼를 오버플로우 시켜 vuln()의 ret주소를 win()으로 바꾸는 과정이 필요하다. 여기에 추가로 win()함수의 파라미터를 조정할 필요가 있다. 첫 번째 파라미터를 0xDEADBEEF로, 두 번째 파라미터를 0xDEADC0DE로 바꾸어야 한다.

함수 내의 변수는 스택에서 ret – sfp – variable1 – variable2 - … 의 과정으로 점점 주소값이 작아지는 방향으로 쌓여간다면 파라미터는 … - parameter2 – parameter1 – ret – sfp 와 같이 ret주소의 위에서 점점 주소값이 높아지는 방향으로 쌓여가게 된다. 이를 이용해서 두 파라미터 값을 바꿔줄 것이다.

|  |
| --- |
|  |
| (parameter2) |
| (parameter1) |
| (ret) |
| ret 4bytes |
| sfp 4bytes |
| BUFFER 108bytes |

왼쪽과 같이 스택을 표현한다면

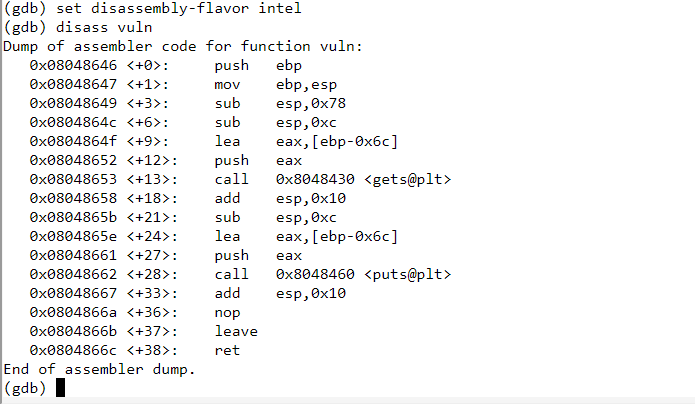
Win()의 ret주소 위를 parameter로 쓸 것이다.

Win()함수가 시작되고 나면 이 곳이 win()함수가 끝날 때 쓸 ret이다.

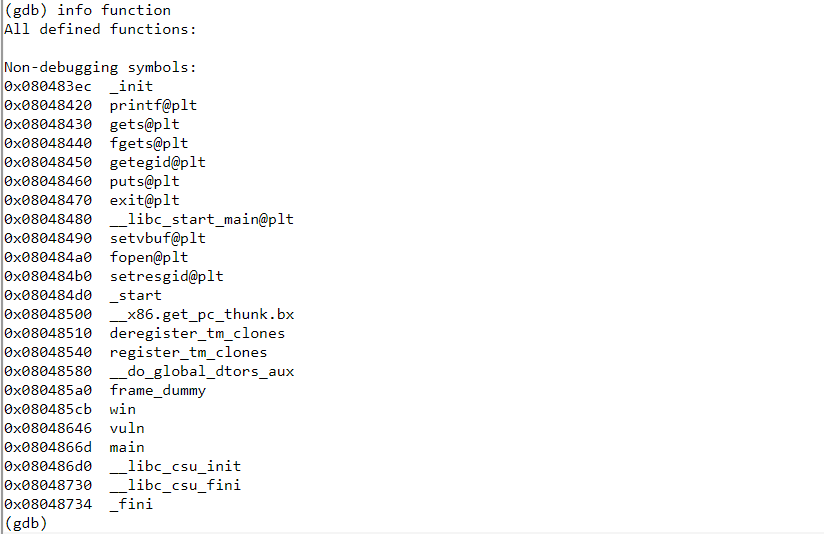
여기의 ret주소를 win()의 주소로 바꾼다.

Sfp는 아무값이나 넣어준다.

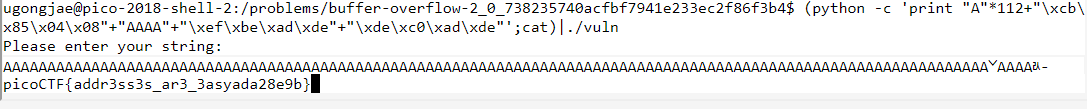
버퍼크기 이상의 값을 넣어서 위의 영역들을 오염시킨다.



위에서 보이듯이 버퍼의 크기는 108바이트이다



win함수의 주소는 0x080485cb로 확인된다.

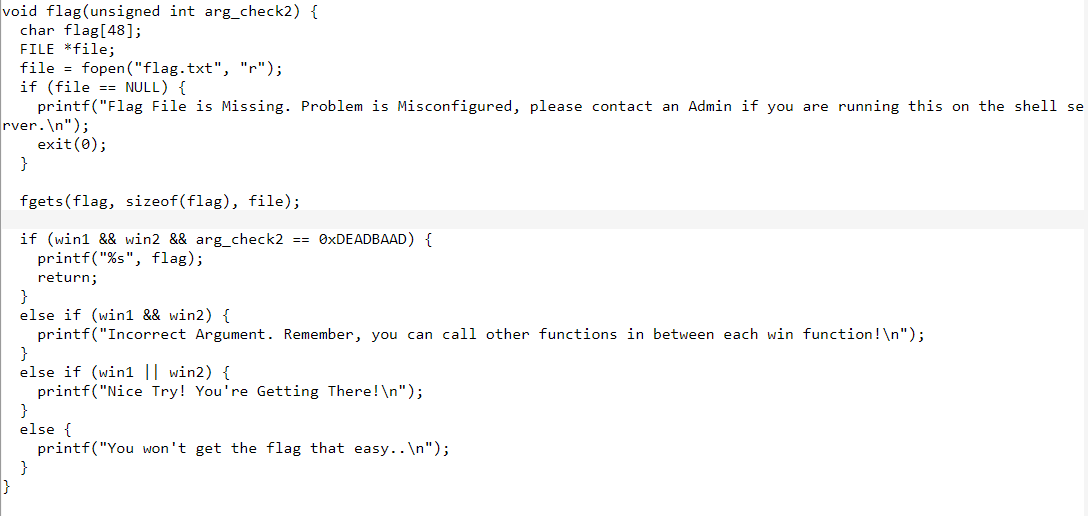


(python –c ‘print “A”\*112+”\xcb\x85\x04\x08”+”AAAA”+”\xef\xbe\xad\xde”+”\xde\xc0\xad\xde”’;cat)|./vuln

위의 코드를 통해 공격할 수 있다.

Flag : picoCTF{addr3ss3s\_ar3\_3asyada28e9b}

1. rop chain



위와 같이 소스코드를 확인해보면 vuln()의 버퍼를 오버플로우 시켜서 win\_function1()의 함수를 통해 win1을 조작하고, 그 후 win\_function2()를 불러와서 win2를 조작하고 중간중간 파라미터를 조작해야 함을 알 수 있다.

