김창현 2017033618

pwn\_admin

첨부된 파일을 checksec 명령어로 보면

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6 | Arch:     amd64-64-little  RELRO:    Partial RELRO  Stack:    No canary found  NX:       NX disabled  PIE:      No PIE (0x400000)  RWX:      Has RWX segments | [cs](http://colorscripter.com/info#e) |

위와 같이 NX나 Canary같은 보호 기법이 적용되지 않았다는 것을 확인할 수 있습니다.

프로그램을 실행시켜 보면

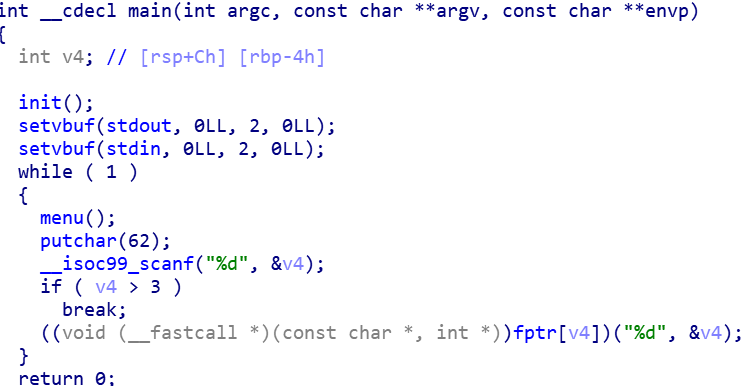
1. Login

2. Logout

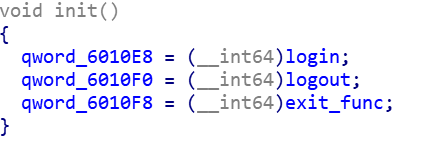
3. Exit

위와 같은 메뉴가 뜨는 것을 확인할 수 있습니다.

IDA 의 Hex ray decompiler로 디컴파일하면 Pseudocode를 볼 수 있습니다.



main함수를 보면 init함수를 호출하는데 이 함수의 내용은 아래와 같습니다.

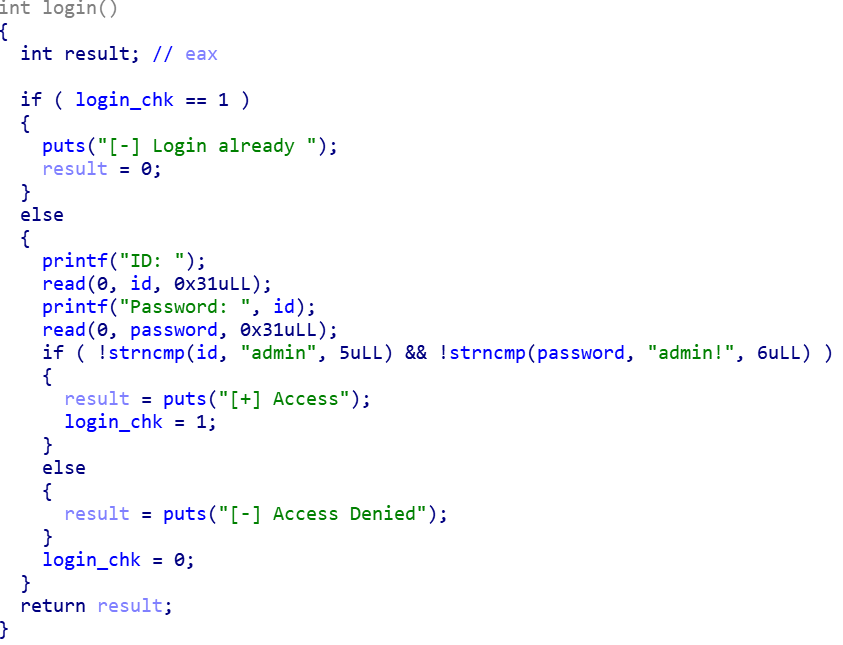


bss영역에 있는 0x6010E8 ~ 0x6010F8에 login , logout, exit\_func함수의 주소를 저장합니다.

다시 main함수로 돌아가 보면 menu함수를 통해 메뉴를 화면에 출력하고 62문자(>)를 출력한 뒤에 정수 하나를 입력 받고 그 값이 3보다 크지 않으면 fptr[v6]을 호출하는데 fptr[1] ~ fptr[3]은 0x6010E8 ~ 0x6010F8이므로 각 번호의 메뉴에 해당하는 함수의 주소가 들어 있습니다.

그런데 음수나 0도 입력할 수 있기 때문에 bss영역에서 0x6010E8 이전 메모리에 어떤 주소를 넣을 수 있다면 그 주소의 코드를 실행할 수 있습니다. 64비트 프로그램이므로 주소가 8바이트이기 때문에 fptr[v4] 는 8바이트 단위로만 설정할 수 있습니다.

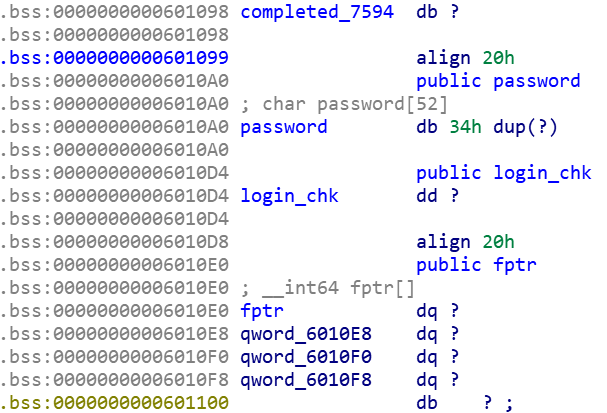
login 함수의 내용은 아래와 같습니다.



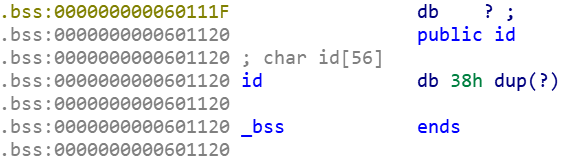
0x31바이트만큼 읽어서 id(bss영역)에 저장하고 0x31바이트 만큼 읽어서 password(bss영역)에 저장합니다.

각각 admin, admin! 과 비교하는데 strncmp함수로 비교하기 때문에 정확히 3번째 인자에 있는 바이트 만큼만 비교합니다. 즉 id에서 admin 이후 또는 password에서 admin! 이후에 추가로 입력해도 로그인이 성공하는 것입니다.

Id와 password가 bss영역에서 어디에 위치하는지 보면 아래와 같습니다.



password는 fptr보다 위에 있고,



Id는 fptr보다 아래에 있습니다.

이제 password 부분에 쉘코드가 있는 주소를 넣고, id 부분에 쉘코드를 넣고, main함수에서 발견한 취약점(음수입력가능)을 통해 쉘을 얻으면 됩니다.

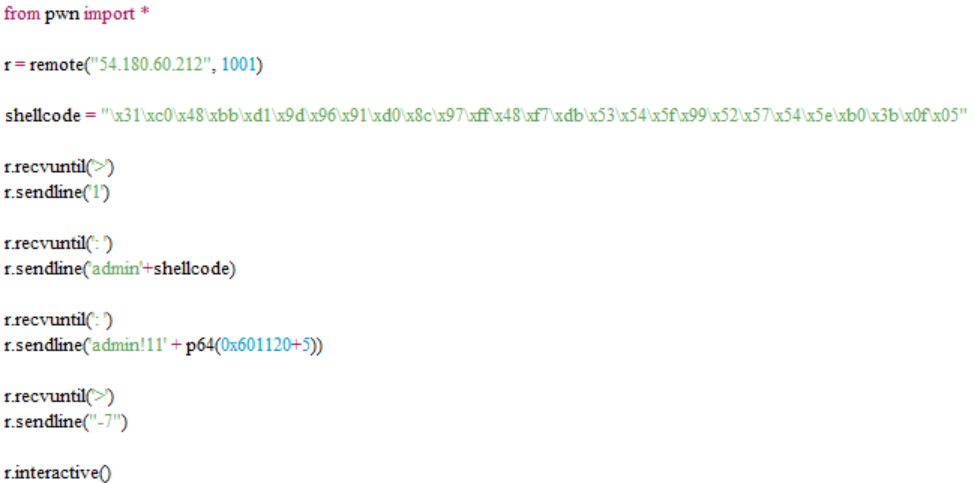
Id의 주소는 0x601120이므로 id에 admin+쉘코드 를 넣는다면

쉘코드는 0x601125에 위치하게 됩니다.

password에 admin!11 + p64(0x601125) 를 입력한 뒤에 (11은 8바이트를 맞추기 위해서 추가했습니다.)

다시 main함수로 돌아와서 (while문이므로)

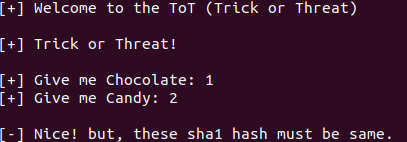
((password주소+8)-fptr) / 8 = -7을 입력해주면 쉘코드가 있는 0x601125로 jmp해서 쉘을 얻을 수 있습니다.



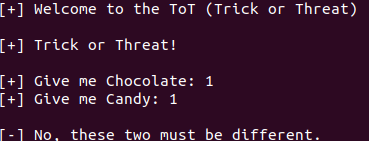
위는 파이썬 코드입니다.

crypto\_ToT

nc로 접속해서 아무 값이나 입력해보면



위와 같이 sha1이 같아야 한다고 뜹니다.



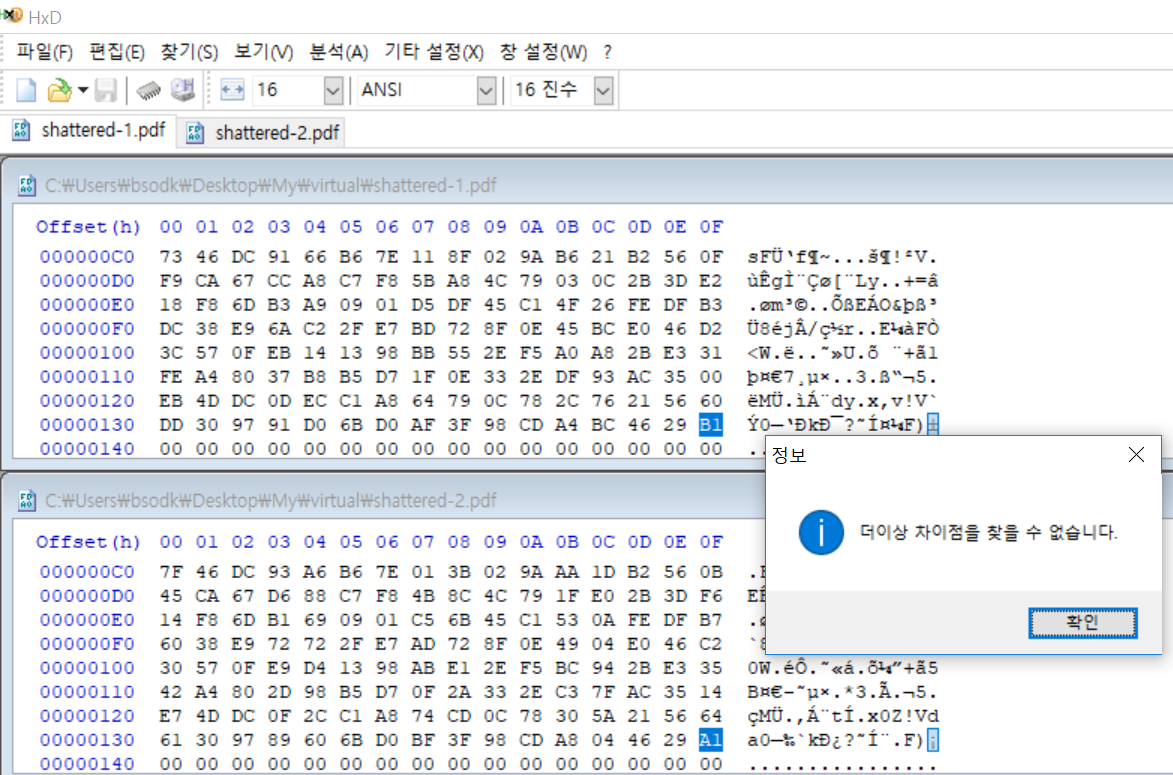
같은 값을 입력하면 두 값이 달라야 한다고 뜹니다.

즉 sha1 hash값의 충돌 쌍을 찾으라는 뜻입니다.

<https://shattered.it/>

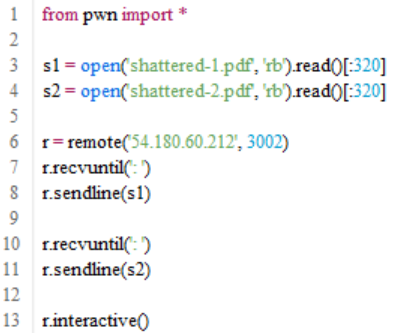
위 주소로 가면 서로 다른 파일이지만 sha1값이 같은 pdf파일을 얻을 수 있습니다.

파일이 매우 크기 때문에 당연히 파일 내용을 전부 입력할 수는 없고 일부만 잘라서 보내야 합니다.



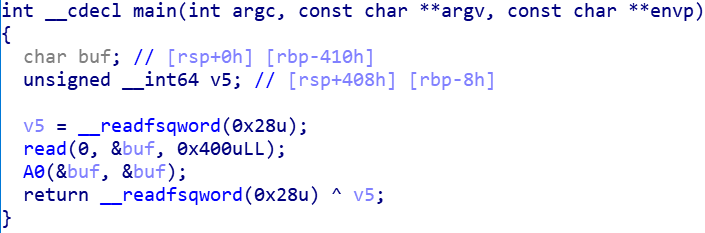
HxD로 파일을 비교해보면 0x13F 이후로 모두 같은 내용이라는 것을 알 수 있습니다.

즉 0~0x13F(319)까지만 보내면 됩니다.



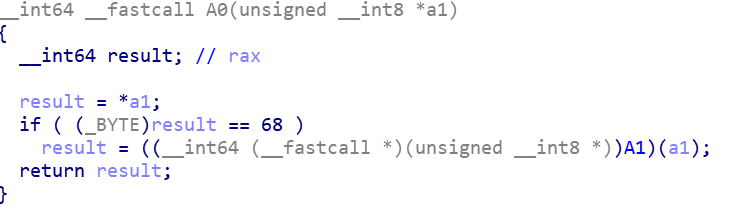
rev\_script

파일을 IDA로 열면

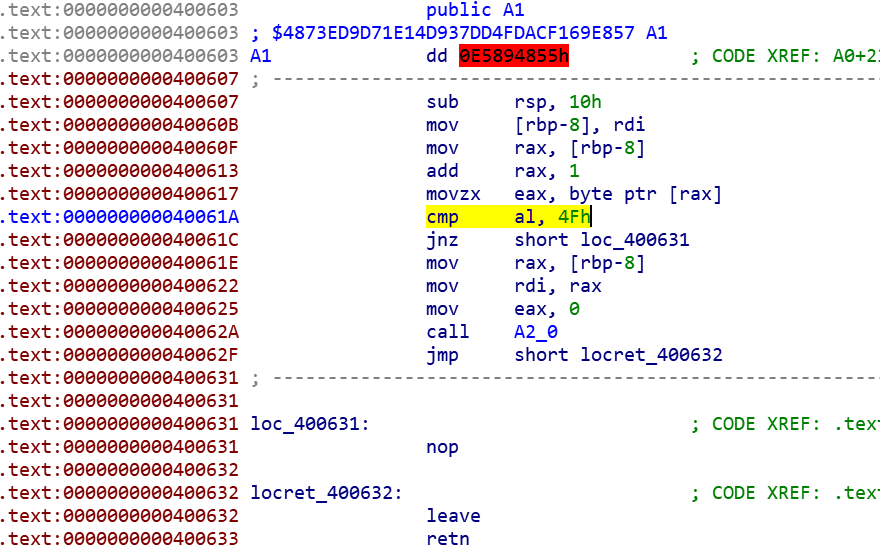


위와 같은 main함수가 있습니다.

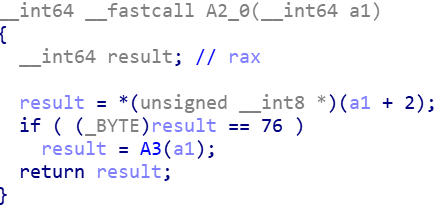
0x400만큼 입력 받은 뒤에 A0함수를 호출합니다.



A0함수는 위와 같은데 \*a1 (=a1[0]=buf[0]) 1바이트를 68과 비교해 같으면 A1을 호출하고 다르면 리턴합니다.

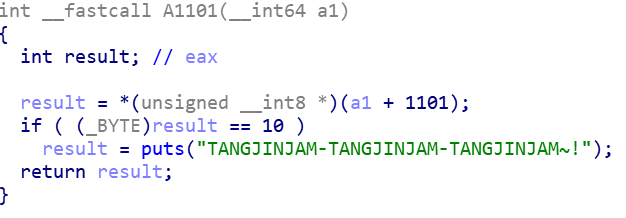


A1은 의사코드로 출력이 안되므로 어셈블리를 보면 [rdi+1] (=buf[1])에 저장된 값 1바이트를 4F와 비교해서 같으면 A2\_0을 호출하고 다르면 리턴합니다.



A2\_0을 보면 a1[2] (=buf[2])를 76과 비교해서 같으면 A3호출, 다르면 리턴합니다.

이러한 함수가 A1101까지 있습니다.

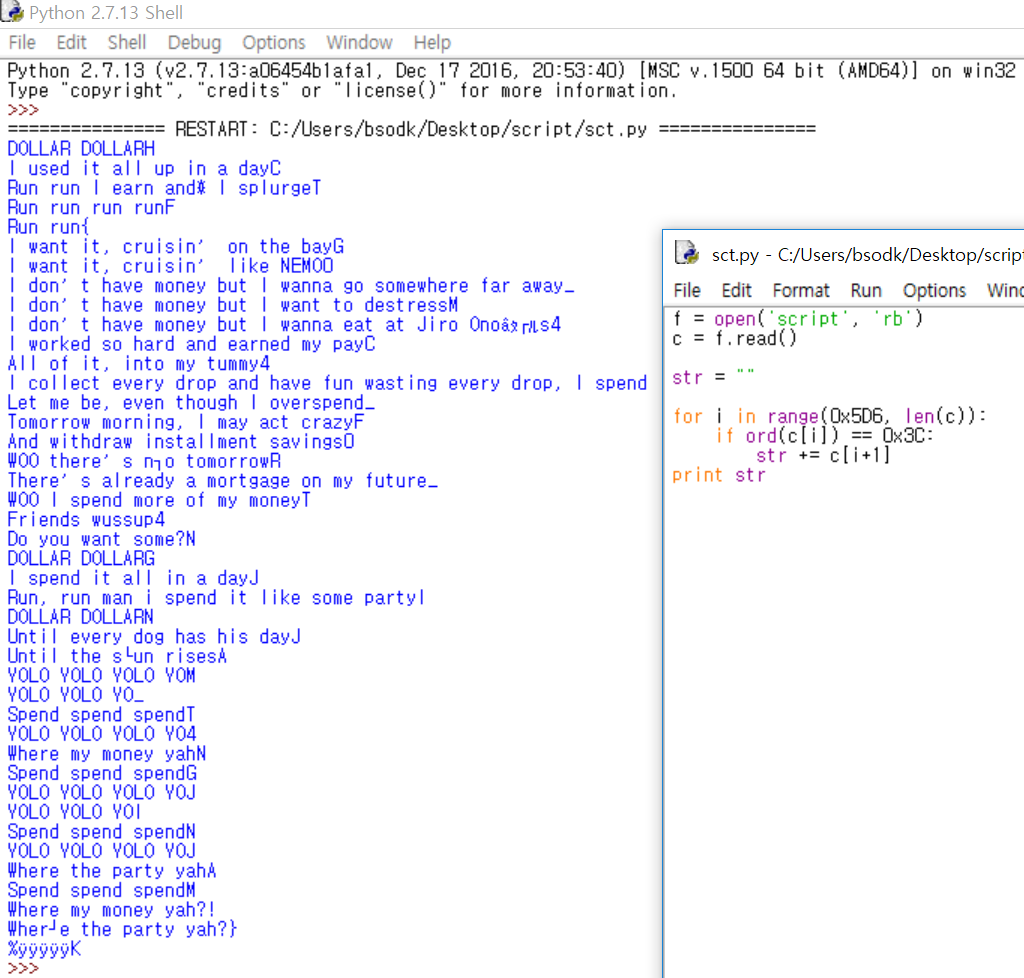


A1101은 10과 비교해서 같으면 TANGJINJAM~ 을 출력합니다.’

이 상태에서 시도해 볼 수 있는 것이 buf를 무슨 문자와 비교하는지 전부 나열해보는 것 밖에 없습니다.

cmp al, ?? 은 3C ?? 이므로 3C를 찾고 그 다음문자를 출력하면 됩니다.

A0함수가 0x5D6에 있고, A1부터는 그 이후에 있으므로 0x5D6부터 끝까지 읽으면 됩니다.



우측 파이썬 코드를 실행하면 왼쪽 결과가 나옵니다.

결과를 자세히 보면 각 줄마다 마지막 문자가 이상하다는 것을 알 수 있습니다.

마지막에 있는 문자 몇 개만 보면 HCTF{ 인데 이것은 플래그의 포맷이라는 것을 알 수 있습니다.

파이썬 코드를 수정해서 \n바로 앞에 있는 문자들만 출력하도록 하면 됩니다.

