[2016 SSG CTF - Break the windows]

바이너리에 심볼이 날아갔지만 바이너리가 작아 큰 문제는 없었다. 문제는 sp가 박살나서 hexray가 안된다는 것이다. 하지만 이것은 sp값을 바꿔줌으로써 쉽게 해결이 가능하다. (크기가 작아 어셈을 보고 분석을 해도 된다.)

다음은 메인함수이다.

```
v0 = sub_4015DC();
setvbuf((FILE *)v0 + 1, 0, 4, 0);
GET_KEY_401170();
sub_401380(&get_string);
dword_417FBC = 0;
puts("==  ADVANCED Memory Corruption Detector. ==");
puts("==   Basic Of Exploitation on Windows.   ==");
dword_417FB4 = TIME(0) & 0xFFFFFFF0;
v8 = &dword_417FB4;
srand(dword_417FB4);
while ( 1 )
{
  PRINT_MENU_4010B0();
  memset(&get_string, 0, 0x64u);
  v1 = RAND_401EB5();
  v2 = RAND_401EB5() * v1;
  v3 = RAND_401EB5();
  v10 = v3 * v2;
  v9 = v3 * v2;
  dword_417FB8 = v3 * v2;
  printf_4019B6("> ");
  scanf_401398("%d", &dword_417FBC);
  v4 = (FILE *)sub_4015DC();
  sub_40176B(v4);
  if ( dword_417FBC != 1 )
    break;
  printf_4019B6("Input your string : ");
  sub_4010F0(&get_string, v7);
  printf_4019B6("This is your string : %s\n", &get_string);
  Sleep(0x64u);
  if ( sub_401020(&dword_417FB8, &v9, 4) != 1 )
  {
    printf_4019B6("[!] Attack Detected.\nBye :pp\n");
    Sleep(0x64u);
    exit(0);
  }
}
if ( dword_417FBC == 2 )
  printf_4019B6("Good bye :p");
else
  puts("Wrong Number..");
Sleep(0x64u);
return 0;
}
```

문자열 변수가 하나 주어지고 그 변수에 계속해서 데이터를 집어넣으면서 공

격을 해보라고 한다. 메모리 릭도 간단하게 일어나서 간단하게 풀 수 있을 것 같다. 하지만 stack cookie가 두 개나 있기 때문에 바로 overflow를 시도하면 프로그램이 종료된다. stack cookie중 하나는 계속해서 rand() 함수를 통해 바뀌기 때문에 메모리 릭으로는 우회할 수가 없다. 하지만 stack cookie는 broken_window 문제처럼 SEH를 이용하면 간단하게 우회가 가능하므로 이것은 문제될 것이 없다.

문자열을 입력받는 함수인 sub_4010F0은 그냥 이뮤니티로 동적 디버깅을 하면서 분석을 했다. 이 함수의 첫 번째 인자는 입력받을 문자열변수, 두 번째 인자는 입력받은 문자열의 크기를 갖고있는 변수이다. 이 변수는 초기값이 0x64로 설정되어있고 문자열 변수의 크기도 0x64이므로 문제가 없어 보이지만 fortune_cookie 문제처럼 등호의 문제로 인해 1바이트가 overwrite가 일어나고 입력받는 문자열의 크기를 조작할 수 있어 overflow가 일어난다.

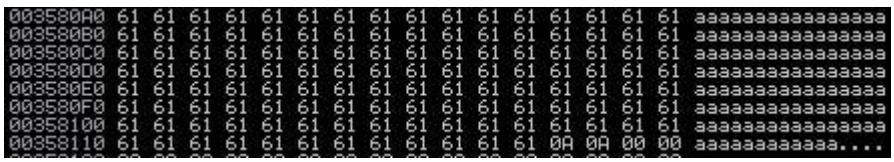
```
012C10FF  >  8B45 F8        MOV EAX,DWORD PTR SS:[EBP-8]
012C1102  .  83C0 01        ADD EAX,1
012C1105  .  8945 F8        MOV DWORD PTR SS:[EBP-8],EAX
012C1108  >  8B4D F8        MOV ECX,DWORD PTR SS:[EBP-8]
012C110B  .  3B4D 0C        CMP ECX,DWORD PTR SS:[EBP+C]
012C110E  .  7F 39          JG SHORT Break_th.012C1149
012C1110  .  E8 C7040000    CALL Break_th.012C15DC
012C1115  .  BA 20000000    MOV EDX,20
012C111A  .  6BCA 00        IMUL ECX,EDX,0
012C111D  .  03C1           ADD EAX,ECX
012C111F  .  50             PUSH EAX
012C1120  .  E8 83070000    CALL Break_th.012C18A8
012C1125  .  83C4 04        ADD ESP,4
012C1128  .  8845 FF        MOV BYTE PTR SS:[EBP-1],AL
012C112B  .  0FBE55 FF      MOVSX EDX,BYTE PTR SS:[EBP-1]
012C112F  .  83FA 0A        CMP EDX,0A
012C1132  .  75 08          JNZ SHORT Break_th.012C113C
012C1134  .  8B45 F8        MOV EAX,DWORD PTR SS:[EBP-8]
012C1137  .  8945 F4        MOV DWORD PTR SS:[EBP-C],EAX
012C113A  .  EB 0D          JMP SHORT Break_th.012C1149
012C113C  >  8B4D 08        MOV ECX,DWORD PTR SS:[EBP+8]
012C113F  .  034D F8        ADD ECX,DWORD PTR SS:[EBP-8]
012C1142  .  8A55 FF        MOV DL,BYTE PTR SS:[EBP-1]
012C1145  .  8811           MOV BYTE PTR DS:[ECX],DL
012C1147  .^ EB B6          JMP SHORT Break_th.012C10FF
012C1149  >  33C0           XOR EAX,EAX
012C114B  .  74 16          JE SHORT Break_th.012C1163
012C114D  .  54             PUSH ESP
012C114E  .  81C4 4C030000  ADD ESP,34C
012C1154  .  8BC4           MOV EAX,ESP
012C1156  .  33E4           XOR ESP,ESP
012C1158  .  FFE4           JMP ESP
012C115A  .  8BE0           MOV ESP,EAX
012C115C  .  81EC 4C030000  SUB ESP,34C
012C1162  .  5C             POP ESP
012C1163  >  8B45 F4        MOV EAX,DWORD PTR SS:[EBP-C]
012C1166  .  8BE5           MOV ESP,EBP
012C1168  .  5D             POP EBP
012C1169  .  C3             RETN
```

위는 sub_4010F0 함수에서 문자열을 한바이트씩 fgetc() 함수를 통해 문자열 변수에 입력하는 루틴이다. 코드를 보면 for(i=0 ; i<=size ; i++) 같은 코드인 것을 쉽게 알 수 있다.

문자열 사이즈 변수 바로 위에 off.17FB4의 주소가 저장되어 있으니 이것을 leak해서 base의 주소를 구할 수 있다.

또한 문자열이 저장되는 메모리 영역을 보면 멀지 않은곳에 SEH를 발견할 수 있다. 계산을 해보니 문자열을 0xb0개 만큼 써넣으면 SEH를 덮어 쓸 수 있다.

nx가 꺼져있으니 쉘코드를 넣을 공간만 찾으면 된다. Exception이 발생했을 때에 edi레지스터가 문자열의 주소를 가지고 있길래 edi값을 esp에 넣는 가젯을 찾으려 했지만 찾지 못했다. 여러 가지 삽질을 하던 도중 fgetc() 함수를 문자열을 어디에서 가져오는지 궁금해져서 동적 디버깅을 하면서 찾아봤다.



찾았다. 마침 data영역이라 base 주소를 알면 쉽게 접근할 수 있다. 이곳에 쉘코드를 넣어놓고 SEH를 이 주소로 바꾼다면 exploit이 가능할 것이다.

< Exploit >

[1byte overwrite] -> [leak off.17FB4 addr] -> [input shellcode] -> [overwrite SEH] -> exploit~!!

(코드는 따로 첨부)