

[2015 Codegate Qual - bookstore2]












프로그램을 실행시키면 id와 pw를 입력하라고 한다. IDA로 해당 부분을 열어 보면 id와 pw는 쉽게 구할 수 있다.

```
signed int LOGIN_401520()
{
    char *v0; // ST0C_4@1
    signed int result; // eax@3
    char *v2; // [sp+4h] [bp-20h]@1
    char v3[4]; // [sp+8h] [bp-1Ch]@1
    char v4[4]; // [sp+14h] [bp-10h]@1

    v0 = (char *)HeapAlloc(dword_4172F0, 8u, 0x14u);
    v2 = (char *)HeapAlloc(dword_4172F0, 8u, 0x14u);
    strcpy(v3, "helloadmin");
    strcpy(v4, "Iulover!@#");
    printf_4027ED((int)"id : ", (char)v0);
    sub_401460((int)v0, 15);
    printf_4027ED((int)"pw : ", (char)v2);
    sub_401460((int)v2, 15);
    if ( !strncmp(v3, v0, 0xAu) && !strncmp(v4, v2, 0xAu) )
    {
        puts("Correct!");
        result = 1;
    }
    else
    {
        result = 0;
    }
    return result;
}
```

id : helloadmin pw : Iulover!@#

함수가 꽤 많으므로 문제풀이에 필요한 함수만 보도록 하자.

-  ADD_NEW_401000
-  sub_401460
-  free_4014A0
-  sub_4014C0
-  LOGIN_401520
-  MODIFY_AVAILABLE_401630
-  MODIFY_PROPERTIES_401740
-  MODIFY_BOOKNAME_4018D0
-  MODIFY_DESCRIPTION_401990
-  MODIFY_401A50
-  MODIFY_MENU_401B60
-  PRINT_DESCRIPTION_401B80
-  MAIN_MENU_401BD0
-  REMOVE_BOOK_401C50
-  PRINT_LIST_401D70
-  sub_401E90
-  VIEW_INFO_401EA0
-  sub_402070
-  MAIN

먼저 새로운 책을 만드는 ADD_NEW 함수이다.

```
puts("Add New Item!Wn1. BookWn2. E-Book");
scanf_4029F1("%d", &type);
v0 = (FILE *)sub_4023CD();
sub_40255C(v0);
```

책에는 일반 Book과 E-Book 두 가지 타입이 있다. 책의 타입에 따라 옵션과 정보를 저장하는 구조체의 정보가 달라진다.

첫 번째 타입인 Book을 보자. 입력받는 정보는 이름, 설명, 재고, 가격, 무료배송 여부 이다. 해당 정보들은 다음과 같이 구조체에 저장된다.

```
N_Book[1] = count;
N_Book[3] = name;
N_Book[7] = description;
N_Book[4] = price;
N_Book[5] = stoke;
N_Book[8] = v18;
*N_Book = PRINT_DESCRIPTION_401BB0;
```

여기서 한 가지 특이한 점은 함수의 포인터가 구조체에 저장되는 것이다. 이 부분을 릭 하거나 덮어써서 eip를 잡을 수 있을 것 같다.

두 번째 타입인 E-Book이 입력받는 정보는 이름, 설명, 재고, 가격, 최대 다운로드 수 이다. 해당 정보는 다음과 같이 구조체에 저장된다.

```
*N_EBook = count;
N_EBook[2] = Ename;
N_EBook[11] = Edescription;
N_EBook[8] = Eprice;
N_EBook[9] = Estoke;
N_EBook[10] = Emax;
```

책의 정보를 구조체에 저장하는 부분을 보면 구조체의 구조를 대충 파악할 수 있다.

다음은 책의 정보를 수정해주는 곳인 MODIFY를 열어보자

```
puts("1. Modify bookname");
puts("2. Modify description");
puts("3. Modify Properties");
puts("4. Modify Available");
return puts("0. back to main menu!");
```

책의 이름, 설명 등 모든 거의 정보를 수정할 수 있다.

다음은 책을 삭제하는 부분인 REMOVE_BOOK 부분을 보자

```
signed int __cdecl REMOVE_BOOK_401C50(int index)
{
    int type; // [sp+0h] [bp-Ch]@1
    LPVOID v3; // [sp+4h] [bp-8h]@8
    LPVOID lpMem; // [sp+8h] [bp-4h]@2

    type = *((_DWORD *)dword_4172F8 + 4 * index + 1);
    if ( type == 1 )
    {
        lpMem = (LPVOID)*((_DWORD *)dword_4172F8 + 4 * index + 3);
        if ( !*((_DWORD *)lpMem + 6) && !*((_DWORD *)dword_4172F8 + 4 * index + 2) )// true if not available
        {
            free_4014A0(lpMem);
            if ( !*((_DWORD *)lpMem + 5) ) // no stoke
            {
                *((_DWORD *)dword_4172F8 + 4 * index + 3) = 0;
                puts("Removed");
                return 1;
            }
        }
    }
    else if ( type == 2 )
    {
        v3 = (LPVOID)*((_DWORD *)dword_4172F8 + 4 * index + 3);
        if ( !*((_DWORD *)v3 + 4) && !*((_DWORD *)dword_4172F8 + 4 * index + 2) )// true if not available
        {
            free_4014A0(v3);
            if ( !*((_DWORD *)v3 + 9) ) // no stoke
            {
                *((_DWORD *)dword_4172F8 + 4 * index + 3) = 0;
                puts("Removed");
                return 1;
            }
        }
    }
    puts("Wrong..");
    return 0;
}
```

available 값이 0으로 설정되어 있다면 free를 해주고 stoke도 0이라면 free 된 부분을 가리키고 있는 포인터를 0으로 초기화시킨다.

만약 available만 0이라면 포인터가 그대로 남아있어 uaf가 일어나게 된다.

<취약점!!>

이 취약점을 이용하면 함수 포인터를 릭하거나 덮어쓰는 공격이 가능하다.

정보가 저장되는 공간을 보면 다음과 같다.

```
00030000 00 00 00 00 01 00 00 00 00 00 00 00 E0 07 39 00 ....0.....79.  
00030010 01 00 00 00 02 00 00 00 01 00 00 00 E0 07 39 00 0...0...0...79.
```

첫 번째 공간에는 책의 no, type, available, 자세한 정보가 있는 구조체의 주소가 들어있다. 위의 상황은 Book을 free한 뒤에 E-Book을 만들어준 뒤의 모습이다. 보다시피 0번 책은 구조체의 포인터를 그대로 가지고 있는 것을 볼 수 있다.

E-Book을 free한 후에 그 자리에 Book을 새로 할당받았다고 가정해 보자. 정상적으로 Book의 index를 통해 정보를 출력하면 Book의 문제가 없지만, free된 E-Book도 여전히 포인터를 가지고 있기 때문에 E-Book의 index를 통해 해당 메모리 공간을 읽어올 수 있다.

E-Book을 통해 읽는 공간에 Book의 함수 포인터가 위치하게 된다면 해당 부분을 읽어 메모리 립이 가능하다. 또한 립된 값으로 base address를 구할 수 있다.

메모리 립을 위해서 E-Book 수 개를 free시킨 후에 Book을 할당시킨 후 하나씩 정보를 출력하면서 출력되는 정보를 확인하면 세 번째로 만든 E-Book의 정보를 읽을 때 출력되는 Price 부분이 E-Book을 모두 free하고 4번째로 만든 Book의 함수 포인터라는 것을 알 수 있다.

```
> 4  
Select Index  
2  
[*] Index : 2  
[*] Type : EBook  
[*] Name : <null>  
[*] Description : <null>[*] Available : 1  
[*] Price : 3021744  
[*] Stoke : 7  
[*] Max Download : 0
```

3번째 E-Book의 Price부분이 4번째 Book의 함수포인터와 겹치므로 3번째 E-Book의 Price를 수정해 주면 eip를 바꿀 수 있다. 이때 available이 0인 책은 수정이 불가하므로 Book을 free해주고 다시 E-Book을 만들어줘야 한다.

dep가 걸려있으므로 rop를 통해 익스플로잇을 해야 한다. 마침 바이너리에 VirtualAlloc() 함수가 있으므로 이 부분을 사용하면 될 것 같다.

```
LPVOID __cdecl sub_402070(LPVOID lpAddress, SIZE_T dwSize, DWORD flProtect)
{
    return VirtualAlloc(lpAddress, dwSize, 0x1000u, flProtect);
}
```

eip를 0x41414141로 바꾼 뒤 레지스터를 확인해 보자.

```
eax=00610860 ebx=00000000 ecx=41414141 edx=00000000 esi=00000000 edi=00000000
eip=41414141 esp=0047fbcc ebp=0047fc00 iopl=0         nv up ei pl nz ac pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010216
41414141 ??                ???
```

eax레지스터가 3번째 E-Book의 Price 부분을 가리키고 있다. 가젯을 찾아 esp에 eax값을 집어넣는다면 esp를 3번째 E-Book의 Price부분으로 옮길 수 있을 것이다.

해당 가젯은 바이너리를 조금만 분석하면 쉽게 찾을 수 있다.

```
.text:00401E97          xchg     eax, esp
.text:00401E98          pop      ecx
.text:00401E99          pop      eax
.text:00401E9A          retn
```

이 가젯을 사용해서 eax값을 esp에 넣어 esp가 heap영역을 가리키게 할 수 있다. 하지만 해당 부분은 값을 쓰는게 한정적이라 많은 값을 쓸 수 있는 다른 부분으로 esp를 옮겨야 한다.

이때 eax가 가리키는 값 뒤에 8바이트만큼 더 수정 가능하므로 eax에 들어갈 값을 esp를 옮길 주소로 놓고 이 가젯을 한번 더 사용하면 esp를 원하는 곳으로 옮길 수 있다.

이제 esp를 옮길 공간을 찾아야 하는데 해당 공간의 주소는 하드코딩되어 들어가므로 변하지 않거나 base주소로 계산 가능한 곳이어야 한다.

다음 함수를 보자.

```
int __cdecl MODIFY_DESCRIPTION_401990(int index)
{
    FILE *v1; // eax@1
    int result; // eax@2
    int type; // [sp+Ch] [bp-4h]@1

    HeapAlloc(dword_4172F0, 8u, 0x12Cu);
    puts("Description?");
    v1 = (FILE *)sub_4023CD();
    sub_402699((int)&unk_4171C0, 300, v1);
    type = *((_DWORD *)dword_4172F8 + 4 * index + 1);
    if ( type == 1 )
    {
        result = *((_DWORD *)dword_4172F8 + 4 * index + 3);
        *((_DWORD *)(result + 28) = &unk_4171C0;
    }
    else if ( type == 2 )
    {
        result = *((_DWORD *)dword_4172F8 + 4 * index + 3);
        *((_DWORD *)((*(_DWORD *)dword_4172F8 + 4 * index + 3) + 44) = &unk_4171C0;
    }
    else
    {
        result = puts("Something went wrong..");
    }
    return result;
}
```

description을 구조체 안에 써넣기 전에 unk_4171C0에 써넣고 해당 부분을 구조체에 써넣는 것을 볼 수 있다. 이 부분에 DEP해제를 위한 ROP 가젯들을 넣어놓고 esp를 옮기면 된다.

DEP 해제는 주어진 VirtualAlloc 함수를 이용하면 된다. 이때 unk_4171C0에 입력받을 때에는 널바이트 등에 상관없이 모두 써 넣으므로 rtl을 이용해 VirtualAlloc을 사용하면 된다.

이제 DEP 해제까지 해결되었으므로 셸코드를 넣을 공간과 그 공간에 셸코드를 복사해 넣을 방법을 찾아야 한다. 여러 부분을 분석해 보았지만 미리 넣어놓을만한 공간을 찾지 못했다. 따라서 입력받는 함수를 이용해 원하는 공간에 값을 써넣기로 했다. 필요한 함수는 인자로 넘겨진 주소에 길이제한 없이 입력받는(또는 길이를 설정 가능한) 함수이다. 이 함수는 LOGIN 부분에서 입력받을 때 사용되는 sub_401460 함수를 사용하면 된다.

이제 익스플로잇을 짜보자.

1. E-Book 4개 생성
2. E-Book 4개 free
3. Book 4개 생성
4. 3번째 E-Book을 통해 주소값 leak
5. 3번째 E-Book을 통해 함수 포인터 overwrite -> stack pivoting
6. unk_4171C0 에 VirtualAlloc + sub_401460 입력
7. 4번째 Book의 함수 포인터를 참조하게 해 exploit
8. 셸코드 삽입 (sub_401460)
9. Exploit~!!

이때 셸코드를 그냥 실행하면 esp가 쓸 수 없는 메모리를 가리키게 되므로 셸 코드 앞부분에 `mov esp,&writeable_Add` 명령어를 넣어줘야 한다.

(라이트업 참고함)

라이트업을 쓰면서 안 사실인데 디버거에서 프로그램을 실행시키면 힙 청크의 헤더에 쓸데없는 값이 많이 붙는다.(16byte) 이것 때문에 메모리 릭할 때 삽질을 엄청나게 했다. (결국은 노가다를 통해 릭에 성공)

하지만 정상적으로 실행했을 때의 헤더의 크기는 디버거에서 확인한 헤더 사이즈 - 8byte 이다. (32bit)