

# GOM Player 2.0.12 (.ASX)

## Stack Overflow Exploit

### Document V0.2

**HACKING GROUP “OVERTIME”**

**mrboo< [bsh7983@gmail.com](mailto:bsh7983@gmail.com) > 2009.01.10**

이 문서는 2009.01.08일자로 [milw0rm](#)에 DATA\_SNIPER께서 등록한 곰플레이어 관련 exploit을 분석한 문서이다.

Study를 목적으로 만들었고 잘못된 분석일 가능성이 너무 많기 때문에 서로 공유는 하되 잘못된 부분은 메일로 지적 바란다.^^

## < 목 차 >

### **1. execute exploit**

### **2. analyze exploit**

### **3. analyze asx file**

### **4. summary**

# 1. execute exploit

우선 분석하기 전에 **milw0rm**에 올라온 exploit을 한번 실행해 본다.

※ 이 취약점은 이전에 **Parvez Anwar**께서 2007년 2월초에 **GOM Player ASX Playlist Buffer Overflow**라는 이름으로 발표를 한 적이 있다. "ref href" 태그를 이용해서 **URI** 링크를 시키는데 **length**체크를 하지 않아 긴 **URI**를 넣음으로써 스택 기반의 버퍼오버플로우가 발생된다.

## 1.1 실행 환경

OS : Microsoft Windows XP Pro V2002 SP2

GOM Version : 2.0.12.3375

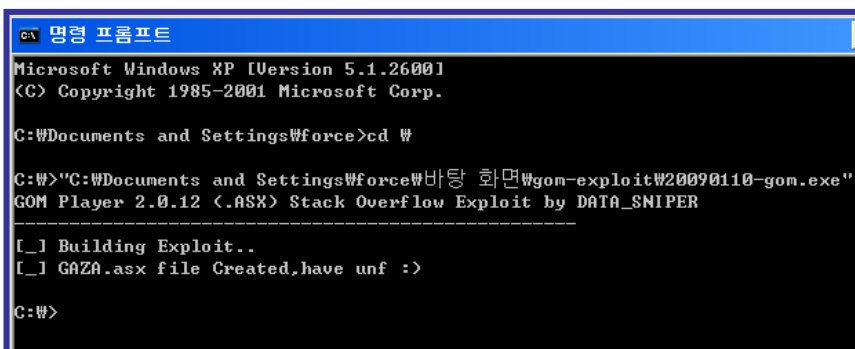
## 1.2 exploit 컴파일

exploit download : <http://milw0rm.com/exploits/7702>

compiler : Visual Studio .NET 2003

## 1.3 바이너리 실행

해당 exploit을 컴파일하고 실행을 한다.



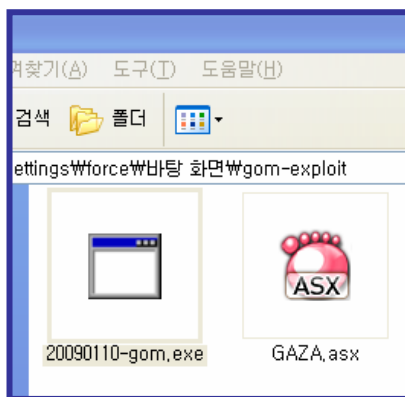
```
명령 프롬프트
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\force>cd #

C:\#>"C:\Documents and Settings\force\바탕 화면\gom-exploit\20090110-gom.exe"
GOM Player 2.0.12 (.ASX) Stack Overflow Exploit by DATA_SNIPER
-----
[_] Building Exploit..
[_] GAZA.asx file Created,have unf :>

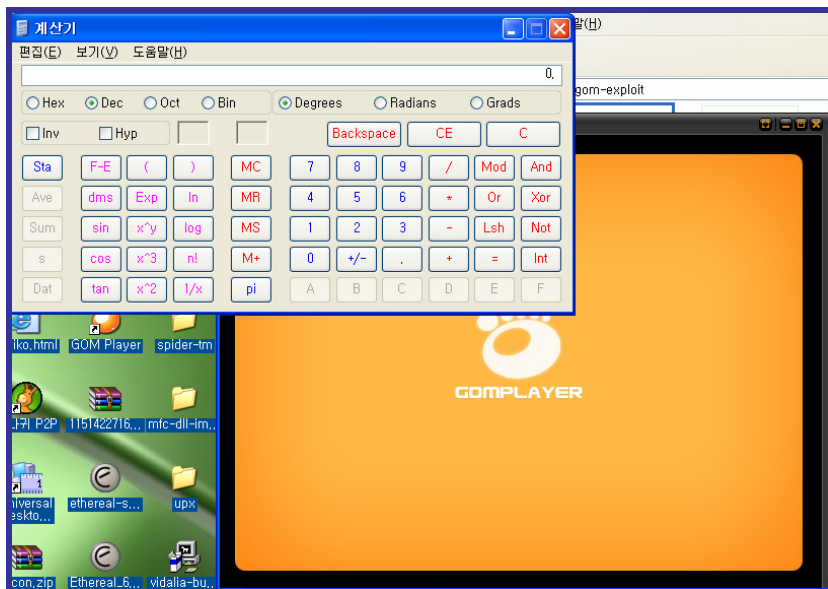
C:\#>
```

같은 위치에 “GAZA.asx”파일이 생성된다.



< 생성된 악성 asx파일 >

“GAZA.asx”파일을 실행시키면 곰플레이어가 실행되면서 계산기가 실행된다.



< 악성 코드로 인해 실행된 계산기 >

## 2. analyze exploit

다음으로 **milw0rm**에 등록된 exploit을 분석해 보자

### 2.1 exploit 분석

*Header1[] : asx파일 포맷의 헤더부분.*

*Header2[] : asx파일 포맷의 끝부분.*

*Shell[] : windows calc(계산기) code.*

*RET\_Univ[] : 0x00464577, 스택오버플로우로 EIP덮어쓰을 주소*

*Nop[] : Nops 8Bytes*

*Payload : 메모리 copy 및 파일 생성*

*위 배열을 Header1+junk(0x41)+RET\_Univ+Nop+Shell+Header2 순으로 payload를 생성 후 GAZA.asx파일에 write함.*

## 3. analyze asx file

생성된 GAZA.asx 파일을 분석해 보자

### 3.1 ASX 란?

#### *ASF Stream Redirector*

실제 동영상 데이터가 아닌 텍스트 정보만을 가지고 있는 메타 파일임.

메모장 등에서 열어 보면 HTML 과 유사한 형식으로 이루어져 있음.

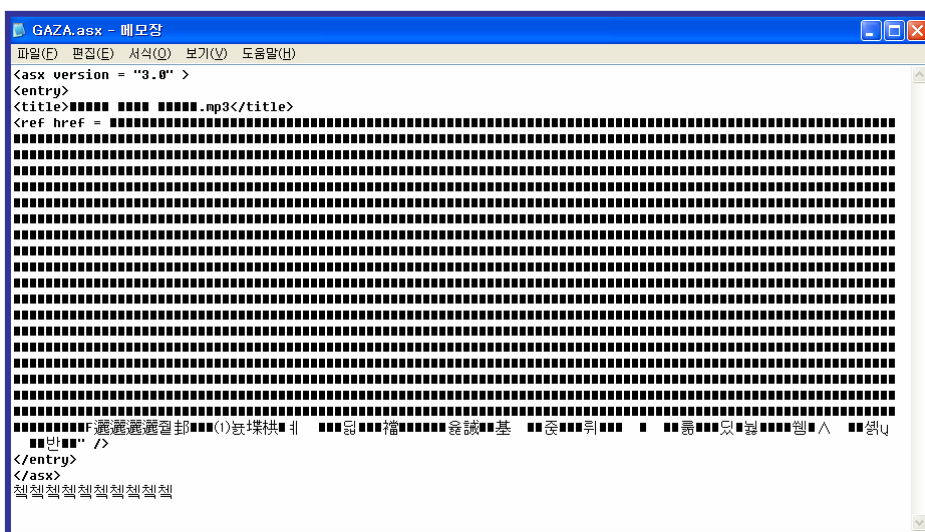
이 파일에는 실제 동영상이 위치하고 있는 URL 정보와 그외의 기타 정보( 저작권, 만드미, 제목 )등을 표기할 수 있고 배너 광고를 삽입하여 자사의 동영상 콘텐츠를 보호 및 홍보할 수 있는 기능을 제공함.

### 3.2 ASX 의 기초

```
<asx version = "3.0" >
  <entry>
    <ref href = "mms://overtimes.com.kr/test.asf"/>
  </entry>
</asx>
```

### 3.2 GAZA.asx 분석

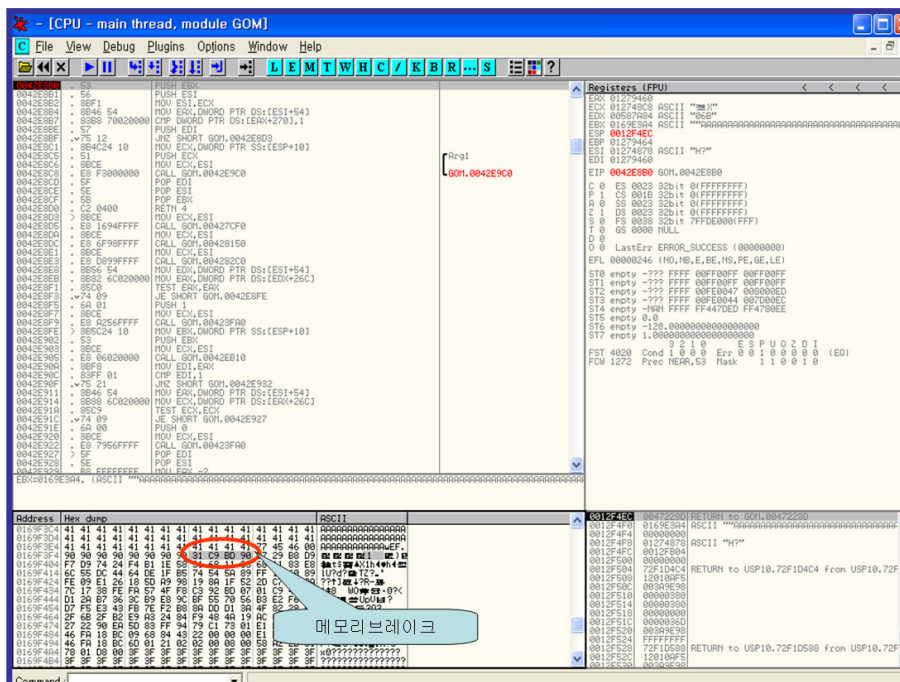
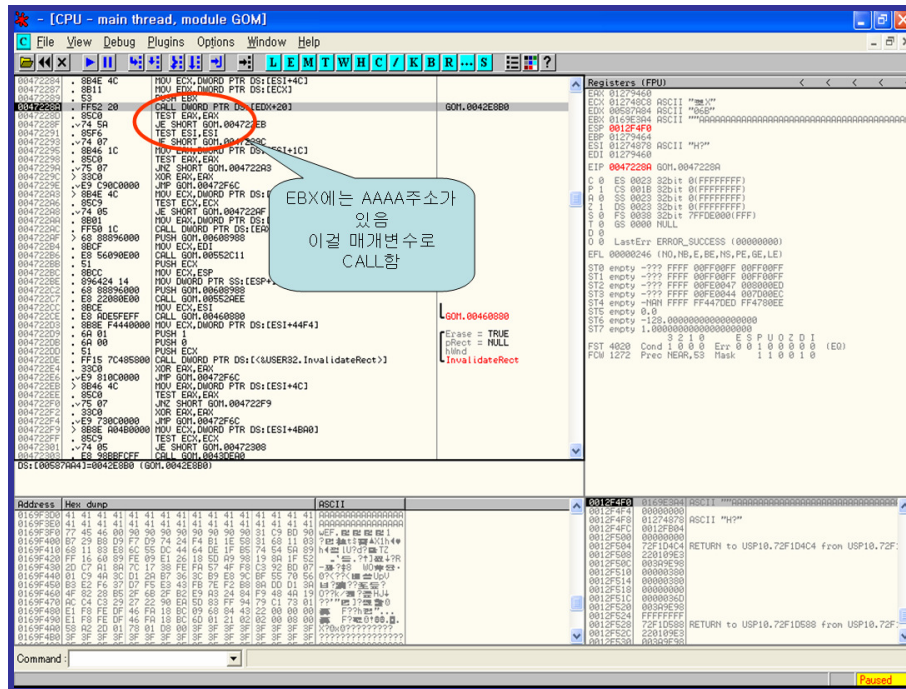
위에서 설명했듯이 GAZA.asx파일을 메모장으로 열어보자



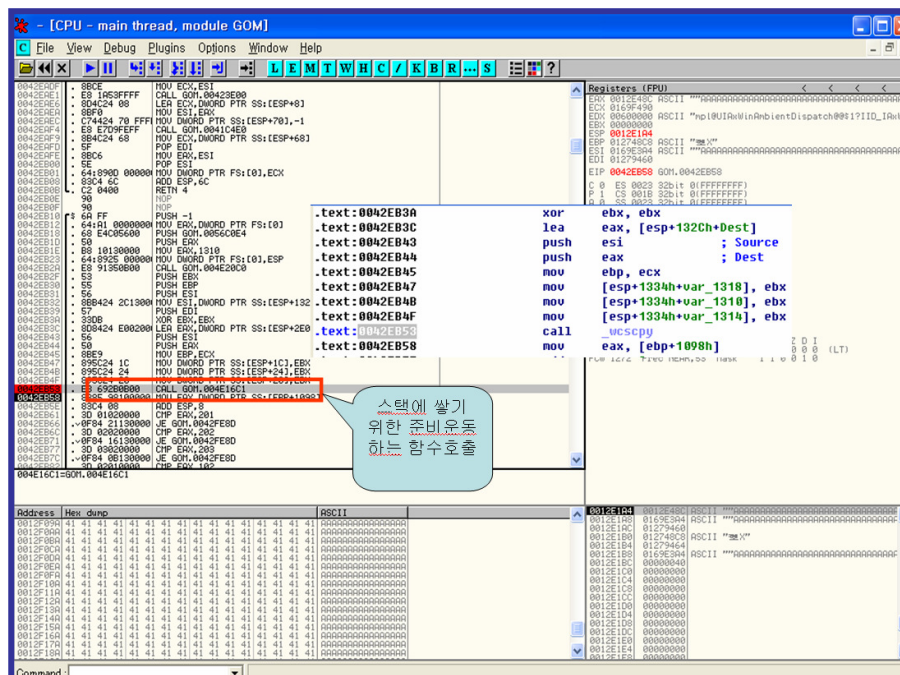
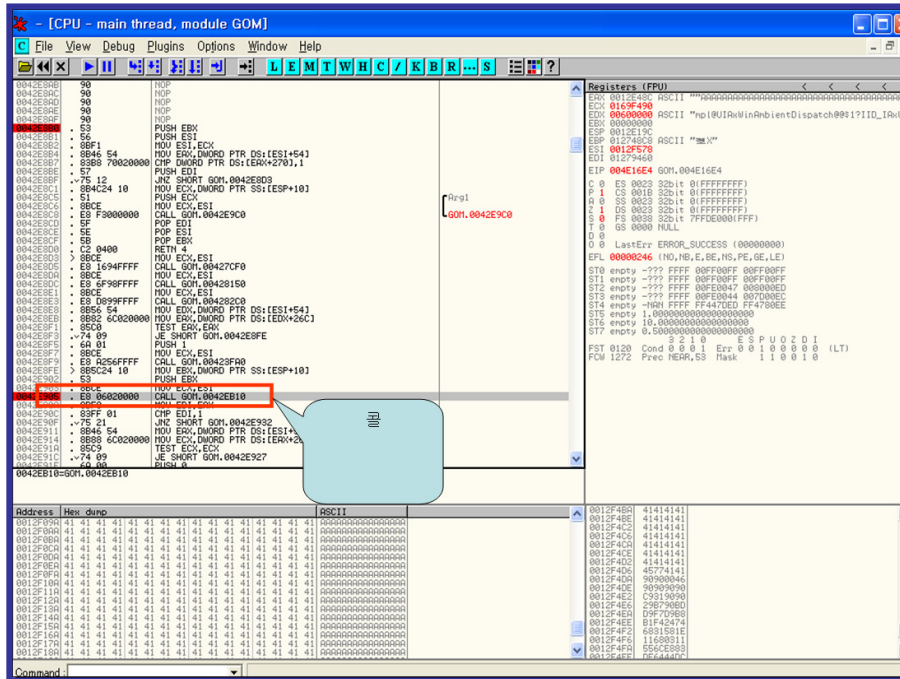
## Ollydbg로 분석해보자

취약한 공플레이어 파일을 올리로 로드한다. 그리고 GAZA.asx 파일을 파일열기로 한 상태에서 브레이크 포인트걸리게 한다. 그럼 메모리에 GAZA.asx파일이 로딩됨을 알 수 있다. **(메모리 서칭기능으로 셀코드의 연속된 부분을 검색어로 서칭하면 찾을 수 있다)** 그리고 해당 메모리에 memory bp를 걸어놔서 이부분을 건드는 코드를 선별한다. 우선 이런식으로 접근해 보자.

step over(F8)로 계속 call문을 지나쳐보면 계산기가 뜨는 시점을 찾을 수 있다. 그럼 해당 call문을 step into(F7)로 자세히 분석해 보자. 수상한 것(?)는 혹시 몰라 캡처를 해놨으며 이 문서에도 그냥 덧붙였다. 별도 설명이 없으면 그냥 넘어가도 될듯하다.







위 그림 중 아래와 같은 CALL문이 눈에 보인다.

**0042EB53** | E8 692B0B00 | CALL GOM.004E16C1

우선 해당 함수가 무슨 일을 하는지 확인해보자

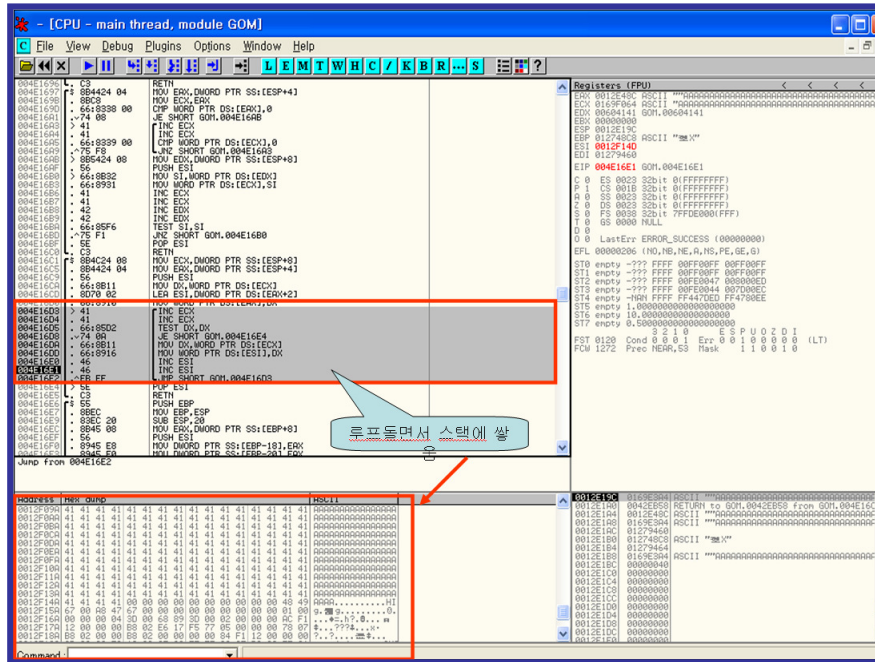
```

004E16D3 > 41      INC EAX
004E16D4 . 41      INC EAX
004E16D5 . 66:85D2 TEST DX,DX
004E16D8 . 74 0A   JE SHORT GOM.004E16E4
004E16DA . 66:8B11 MOV DX,WORD PTR DS:[ECX]
004E16DD . 66:8916 MOV WORD PTR DS:[ESI],DX
004E16E0 . 46     INC ESI
004E16E1 . 4E     INC EDI
004E16E2 ^EB EF   JMP SHORT GOM.004E16D3

```

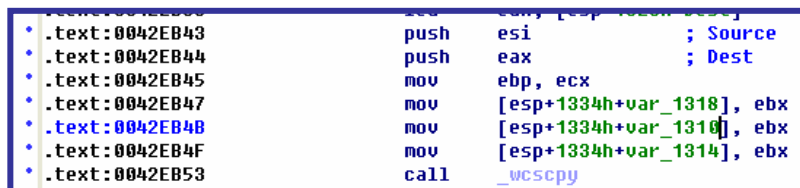


루프문이다. 해당 DS:[ESI]부분을 dump해보니 아래와 같다.



메모리에 있는 asx파일 내용을 메인 스레드의 Stack영역으로 복사하는 것을 볼 수 있다. 복사하는 내용은 asx파일 내용중 "<ref href =" 태그 다음에 오는 실제 동영상 파일의 주소이다. 현재는 쉘코드이다.

IDA에서 분석을 하면 wcsncpy라는 심볼명을 가진 함수임을 알 수 있다.



<IDA로 해당 심볼 확인, wcsncpy>

wcsncpy는 어떤 함수일까? strcpy의 유니코드 버전이라고 보면 된다고 한다.

파일 I/O나 문자열 처리시 기본적으로 2Bytes로 처리를 해버리게 되서 사용한다고 한다.

그런데 이 함수는 보안에 취약하다. 복사할 사이즈를 정해주는 파라미터가 없으며, wcsncpy(dst, src)중 src가 null로 종료를 하지 않으면 그대로 복사가 되고 버퍼가 플로우가 되게 된다. 즉 "href="다음의 주소를 length체크를 하지 않고 복사하기 때문에 우리의 쉘코드가 오버플로우 되면서 로딩 될수 있었던 거다. 이 함수 대신 사용할수 있는 wcsncpy\_s(dst, dst\_size, src) 함수가 있다고 한다.

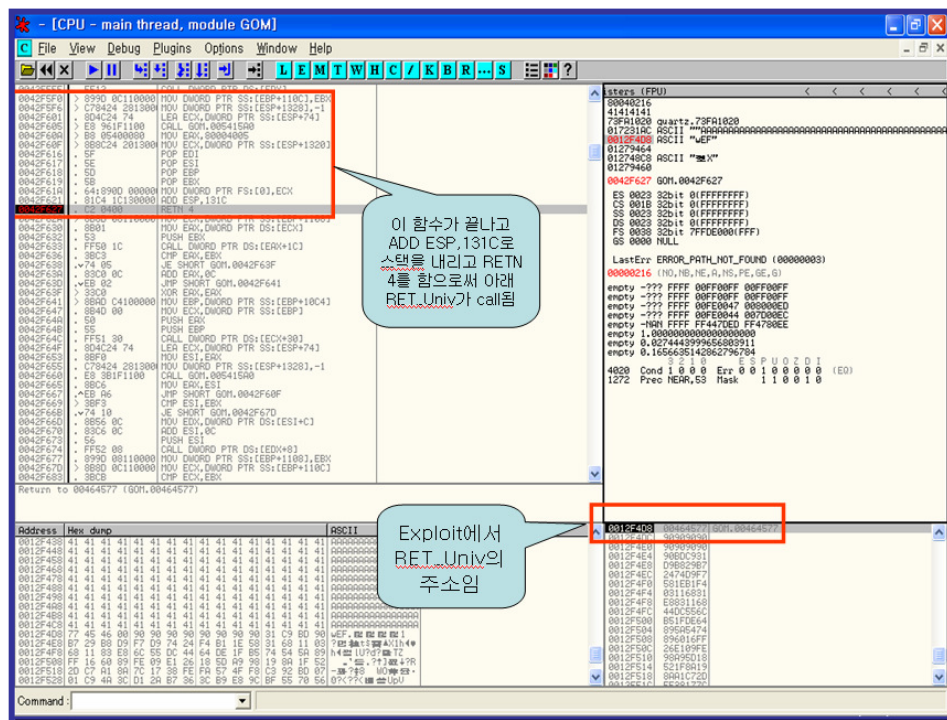
양튼 우리는 양껏 쉘코드를 스택에 넣을 수 있게 됐다.

이제 우리는 스택의 쉘코드를 실행시켜 줄 EIP를 변조시키면 된다.

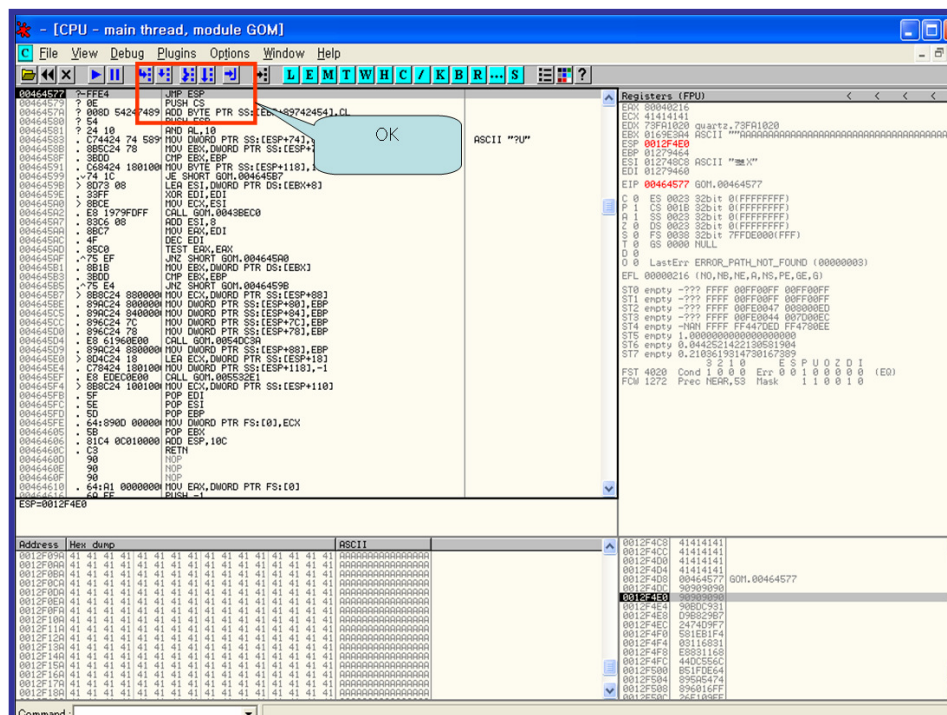
계속 분석해보면 위 wcsncpy를 해놓고 EAX에 SS:[EBP+1098]이 가리키는 값을 EAX에 넣는다. 여기서는 FFFFFFFF가 되겠다. 이 값을 201,202,203,102,103,401,303,601,311,312,901,701,702,101,301,501,801 값과 비교를 하고는 아래 그림부분에 가게 된다. 위 값을 잘은 모르겠으나 정상적인 파일을 올리면 202가 됨을 볼수 있다.

아마 에러처리부분?? 꼭 웹 에러번호처럼...

그럼 계속 분석해보자. 위 FFFFFFFF값으로 어느것 하나 일치가 안되고는 아래 그림으로 넘어가게 된다. RET\_Univ가 기억날 것이다. 기억이 안나면 위 “2. analyze exploit”를 참고



즉 위 그림에 설명처럼 RETN4를 함으로써 RET\_Univ의 주소(0x00464577)가 EIP에 저장되고 실행 된다. 해당 주소변지에는 **JMP ESP** opcode가 있다.



그럼 ESP로 점프하게 되고 ESP에 있는 우리의 NOP코드를 확인할 수 있다. 짧은 NOP 슬레이드를 타시고 바로 calc 코드를 실행하여 계산기가 실행이 된다.

## 4. summary

간단하게 버퍼오버플로우에 취약한 함수로 생기는 홀이 있었으며 공플레이어가 이런 함수를 사용함으로써 exploit이 생겨 난 것임. 내 생각엔.....

언제봐도 이런 취약점을 발견하신 분은 참 대단한 거 같다.

어떻게 이런 생각을 하지?????? 아니 어떻게 이렇게 짜고치는 고스톱처럼 서로 상관없을 듯한, 서로 멀리 떨어진 듯한 코드들이 이렇게 연관성있게 실행되게끔 하는건지 참.....대단한 거 같다.

얌튼 오늘도 그냥 교수분들 대단하단 거 느끼며 문서를 마칠까 한다;;

작성한 문서 내용은 제 지식 위주로 하여 틀린 부분이 많을 수 있다는 위험이 있단 사실을 명심하시고 지적 바랍니다. 마지막으로 패치 후에 코드가 어떻게 변했는지 한번 확인해보자.

감사합니다.

```
.text:0042EB10      push     0FFFFFFFh
.text:0042EB12      mov      eax, large fs:0
.text:0042EB18      push     offset unk_56C0E4
.text:0042EB1D      push     eax
.text:0042EB1E      mov      eax, 1310h
.text:0042EB23      mov      large fs:0, esp
.text:0042EB2A      call     __alloca_probe
.text:0042EB2F      push     ebx
.text:0042EB30      push     ebp
.text:0042EB31      push     esi
.text:0042EB32      mov      esi, [esp+1328h+Source]
.text:0042EB39      push     edi
.text:0042EB3A      xor      ebx, ebx
.text:0042EB3C      lea      eax, [esp+132Ch+Dest]
.text:0042EB43      push     esi          ; Source
.text:0042EB44      push     eax          ; Dest
.text:0042EB45      mov      ebp, ecx
.text:0042EB47      mov      [esp+1334h+var_1318], ebx
.text:0042EB48      mov      [esp+1334h+var_1310], ebx
.text:0042EB4F      mov      [esp+1334h+var_1314], ebx
.text:0042EB53      call     _wcsncpy
.text:0042EB58      mov      eax, [ebp+1098h]
```

<패치 전>

```
.text:004036FE      add      esp, 4
.text:00403701      mov      [esp+288h+var_29C], eax
.text:00403705      cmp      eax, ebx
.text:00403707      mov      byte ptr [esp+288h+var_4], 1
.text:0040370F      jz       short loc_403762
.text:00403711      push     esi
.text:00403712      mov      ecx, edi
.text:00403714      call     sub_402C10
.text:00403719      mov      [esp+288h+h], eax
.text:0040371D      mov      eax, [esp+288h+var_29C]
.text:00403721      mov      [eax+10h], ebx
.text:00403724      push     ebp          ; Str
.text:00403725      mov      [eax+14h], ebx
.text:00403728      call     wcslen
.text:0040372D      lea      eax, [eax+eax*2]
.text:00403731      push     eax          ; Size
.text:00403732      call     sub_57394A
.text:00403737      mov      ecx, [esp+2C0h+var_29C]
.text:00403738      push     ebp          ; Source
.text:0040373C      push     eax          ; Dest
.text:0040373D      mov      [ecx+4], eax
.text:00403740      call     wcsncpy
.text:00403745      mov      eax, [esp+2C8h+var_29C]
```

<패치 후>

ebp에는 SourceString이 들어간다.  
이 함수는 SourceString의 NULL종료문자를 만날 때까지 그 문자열 개수를 세는 함수이다. 그렇기 때문에 SourceString 인자가 NULL종료문자로 끝나지 않는다면, 잘못된 Length를 구하던지, 아니면 할당된 메모리 경계를 벗어나 INVALID한 메모리영역에 침범 시스템 오류를 발생시킬 수 있을 것이다.