Abysssec Research

1) Advisory information

Title	: Java CMM readMabCurveData stack overflow
Version	: Java runtime <= 6.19
Analysis	: http://www.abysssec.com
Vendor	: http://www.java.com
Impact	: Critical
Contact	: shahin [at] abysssec.com , info [at] abysssec.com
Twitter	: @abysssec
CVE	: CVE-2010-0838

2) Not vulnerable version

Sun JRE (Windows Production Release) 1.6.0_19
Sun JRE (Solaris Production Release) 1.6.0_19
Sun JRE (Linux Production Release) 1.6.0_19
Sun JDK (Windows Production Release) 1.6.0_19
Sun JDK (Windows Production Release) 1.5.0_24
Sun JDK (Solaris Production Release) 1.6.0_19
Sun JDK (Solaris Production Release) 1.5.0_24
Sun JDK (Linux Production Release) 1.6.0_19
Sun JDK (Linux Production Release) 1.5.0_24
IBM Java SE 5.0 SR11 PF1
HP Systems Insight Manager 6.1

3) Vulnerability information

Class

1- Stack overflow

Impact

Successfully exploiting this issue allows remote attackers to execute arbitrary code in the context of vulnerable application or cause denial-of-service conditions.

Remotely Exploitable

Yes

Locally Exploitable

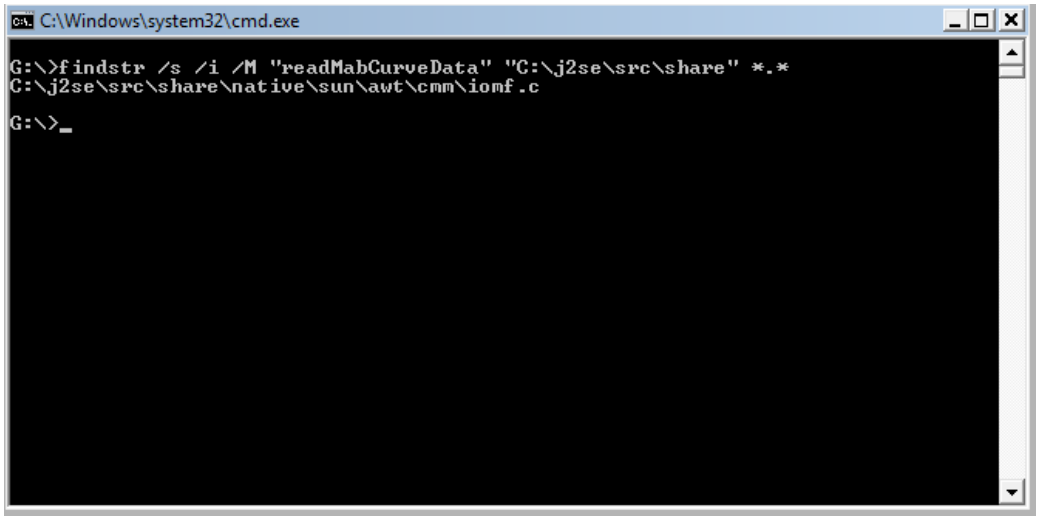
Yes

4) Vulnerabilities detail

The vulnerability exists in readMabCurveData function of CMM module. In order to better understand this function and its usage, it is better to examine java source. You can download java update 18 source from http://dlc.sun.com.edgesuite.net/jdk6/6u18/promoted/b07/index.html.

In path \j2se\src\share you have two important folder named class and native. In native folder there are some source codes written in c or c++. On the other hand in classes folder, java source codes exist. In fact some native function have implemented in native folder similar to their java implementation. The implementation of classes and objects which is used in java are implemented by native c source codes. Although; It is possible for name variations between java and c source codes.

In path \j2se\src\share\native\sun\awt there is folder named cmm and if you search expression readMabCurveData in this path, you can find it in only a unique file and it will be \j2se\src\share\native\sun\awt \cmm\iomf.c .



Well, we have found vulnerable function in the native code. Take a look at the function"

```
static KpInt32_t
readMabCurveData(KpFd_p fd, KpUInt32_t nChan, KpUInt32_p TblEntriesPtr, mab_tbldat_p *TablePtr, PTParaCurve_p
PTParaCurve)
{
mcurve_tcurveType;
KpInt32_tnSig, nTblEntries, nTotalEntries, nTblSize, startOfCurves;
KpUInt16_ttmpTbl [MF2_MAX_TBL_ENT];
KpInt32_tstatus, cOffset;
KpUInt32_t     i1;
KpUInt8_tdummy;
Kp_get_position (fd, &startOfCurves);
nTblEntries = 0;

...
nSig = curveType.nSig;
#if (FUT_MSBF == 0)
       Kp_swab32 ((KpGenericPtr_t)&nSig, 1);
#endif
PTParaCurve[i1].nSig = nSig;
if (CURVE_TYPE_SIG == nSig)
{
        nTblEntries = curveType.C.Curve.nCount;
       #if (FUT_MSBF == 0)
              Kp_swab32 ((KpGenericPtr_t)&nTblEntries, 1);
       #endif
        nTblSize = nTblEntries * sizeof (mab_tbldat_t); /* size in bytes of each table */
        status = Kp_read (fd, (KpGenericPtr_t)tmpTbl, nTblSize);            /* read the input table */
        if (status != 1) {
              return (status);
             }
...
```

As you see in the above code, Kp_read function read content of fd for nTblSize and store it in buffer tmpTbl. Ther flaw here is lack of control on value of nTblSize before using. So it can cause memory corruption.

The vulnerable function is compared by c source code. Now we take a look at it in cmm.dll and compare it with the patched function. Our examinations show that sub_6D185C75 function is equal to the our vulnerable readMabCurveData function.

Bye comparing readMabCurveData and the assembly code of sub_6d185c75 we conclude that the assembly code of calling Kp_read which cause stack overflow is as follow:

```
6D185EBB    MOV EAX,DWORD PTR SS:[EBP-14]
6D185EBE    PUSH EAX
6D185EBF    MOV WORD PTR DS:[EBX-4],AX
6D185EC3    CALL cmm.6D18A0F5
6D185EC8    MOV DWORD PTR SS:[EBP-28],EAX
6D185ECB    SHL EAX,2
6D185ECE    PUSH EAX
6D185ECF    PUSH EBX
6D185ED0    PUSH DWORD PTR SS:[EBP+8]
6D185ED3    CALL cmm.6D1877E2        ---------->        Kp_read (fd, (KpGenericPtr_t)tmpTbl, nTblSize);
```

As you see in the code value from ([ebp-14]) is copied to EAX register and then a little later it will be multiplied by 2 and is passed as the size argument to Kp_read function. there is no control on the passed argument and can corrupt memory:

```
6D185ED2        mov    eax, [ebp+var_18]
6D185ED5        push   eax
```

```
6D185ED6    mov     [ebx-4], ax
6D185EDA    call    sub_6D18A162
6D185EDF    mov     ebx, eax
6D185EE1    add     esp, 0Ch
6D185EE4    test    ebx, ebx
6D185EE6    jl      loc_6D185F77
6D185EEC    cmp     ebx, 7
6D185EEF    jg      loc_6D185F77
6D185EF5    shl     eax, 2
6D185EF8    push    eax
6D185EF9    push    [ebp+lpBuffer]
6D185EFC    push    [ebp+arg_0]
6D185EFF    call    sub_6D18784F   ---------->   Kp_read (fd, (KpGenericPtr_t)tmpTbl, nTblSize);
```