Kaspersky Antivirus Library RemØte Heap Overflow Security Advisory

Date
October 2, 2005

Vulnerability
The Kaspersky Antivirus Library provides file format support for virus analysis. During analysis of cab files Kaspersky is vulnerable to a heap overflow allowing attackers complete control of the system(s) being protected. This vulnerability can be exploited remotely without user interaction in default configurations through common protocols such as SMTP, SMB, HTTP, and FTP.

Impact
Successful exploitation of Kaspersky protected systems allows attackers unauthorized control of data and related privileges. It also provides leverage for further network compromise. Kaspersky Antivirus Library implementations are likely vulnerable in their default configuration.

Affected Products
Due to the library's OS independent design and core functionality: it is likely this vulnerability affects a substantial portion of Kaspersky's gateway, server, and client antivirus enabled product lines on most platforms.

http://www.kaspersky.com/products

Note: Kaspersky's antivirus OEM product line is a program where vendors may license the vulnerable library. The following link is a list containing some of the Kaspersky partners with products also likely affected by this vulnerability. Refer to your vendor for specifics.

http://www.kaspersky.com/oemsuccess

Credit
This vulnerability was discovered and researched by Alex Wheeler.

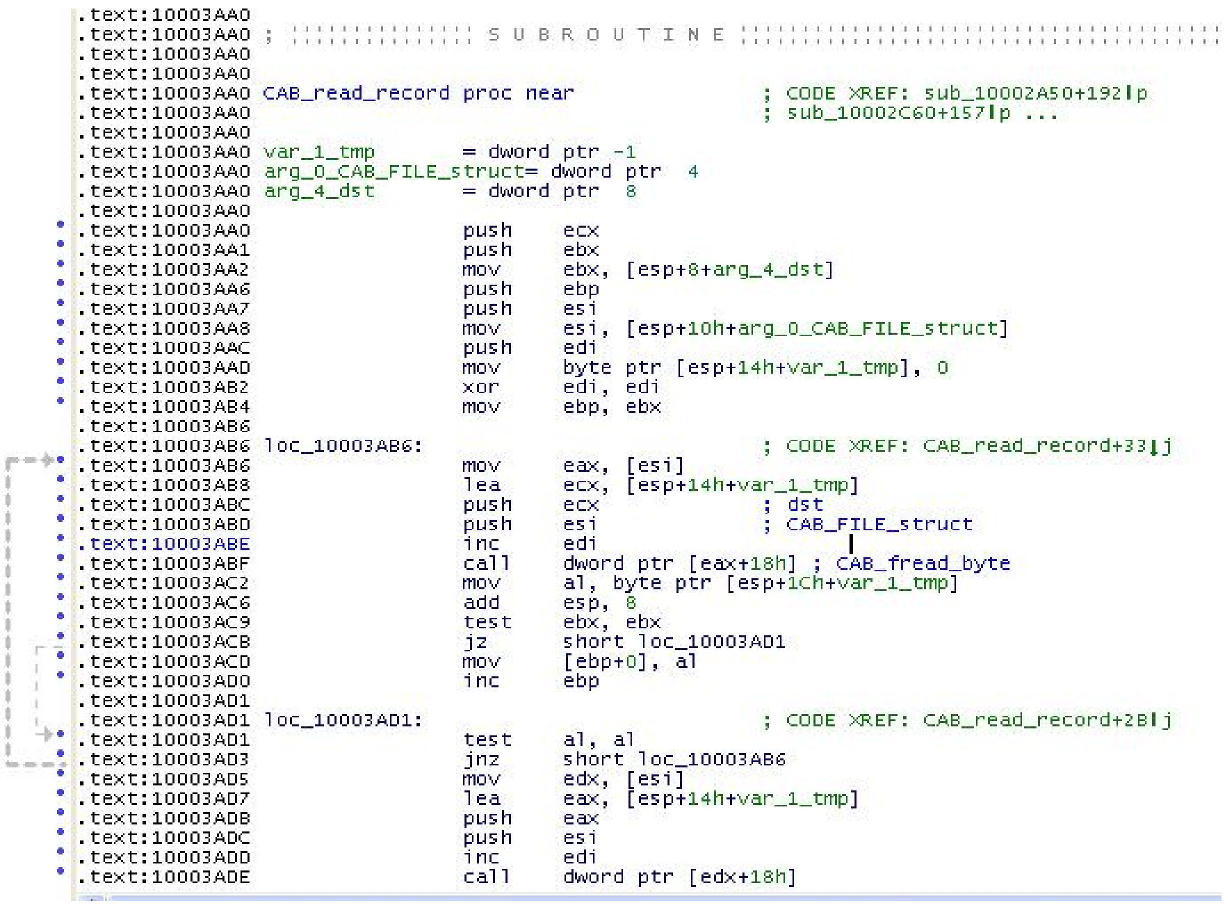
Contact
security@remØte.com

Kaspersky Antivirus Library RemØte Heap Overflow
Security Advisory

Description
The vulnerable file format engine is responsible for parsing cab files. Specifically, the vulnerability is the result of an improperly bounded copy loop in a core processing function.

This function is reachable while processing records after the initial cab header. For many types of records this function is passed a statically allocated heap buffer. By crafting a cab file with large non-null records and particular header flags set, an attacker can corrupt vtables to execute arbitrary machine instructions.

The following vulnerable code is from the cab.ppl file (current at the time of this writing - v5.0.20.0):



```
.text:10003AA0
.text:10003AA0 ; ::::::::::::::: S U B R O U T I N E :::::::::::::::::::::::::::::::::::::::
.text:10003AA0
.text:10003AA0
.text:10003AA0 CAB_read_record proc near               ; CODE XREF: sub_10002A50+192↑p
.text:10003AA0                                         ; sub_10002C60+157↑p ...
.text:10003AA0
.text:10003AA0 var_1_tmp        = dword ptr -1
.text:10003AA0 arg_0_CAB_FILE_struct= dword ptr  4
.text:10003AA0 arg_4_dst        = dword ptr  8
.text:10003AA0
.text:10003AA0                  push    ecx
.text:10003AA1                  push    ebx
.text:10003AA2                  mov     ebx, [esp+8+arg_4_dst]
.text:10003AA6                  push    ebp
.text:10003AA7                  push    esi
.text:10003AA8                  mov     esi, [esp+10h+arg_0_CAB_FILE_struct]
.text:10003AAC                  push    edi
.text:10003AAD                  mov     byte ptr [esp+14h+var_1_tmp], 0
.text:10003AB2                  xor     edi, edi
.text:10003AB4                  mov     ebp, ebx
.text:10003AB6
.text:10003AB6 loc_10003AB6:                           ; CODE XREF: CAB_read_record+33↓j
.text:10003AB6                  mov     eax, [esi]
.text:10003AB8                  lea     ecx, [esp+14h+var_1_tmp]
.text:10003ABC                  push    ecx             ; dst
.text:10003ABD                  push    esi             ; CAB_FILE_struct
.text:10003ABE                  inc     edi
.text:10003ABF                  call    dword ptr [eax+18h] ; CAB_fread_byte
.text:10003AC2                  mov     al, byte ptr [esp+1Ch+var_1_tmp]
.text:10003AC6                  add     esp, 8
.text:10003AC9                  test    ebx, ebx
.text:10003ACB                  jz      short loc_10003AD1
.text:10003ACD                  mov     [ebp+0], al
.text:10003AD0                  inc     ebp
.text:10003AD1
.text:10003AD1 loc_10003AD1:                           ; CODE XREF: CAB_read_record+2B↑j
.text:10003AD1                  test    al, al
.text:10003AD3                  jnz     short loc_10003AB6
.text:10003AD5                  mov     edx, [esi]
.text:10003AD7                  lea     eax, [esp+14h+var_1_tmp]
.text:10003ADB                  push    eax
.text:10003ADC                  push    esi
.text:10003ADD                  inc     edi
.text:10003ADE                  call    dword ptr [edx+18h]
```

The disassembly above approximates to the following source.

```
static int CAB_read_record(CAB_FILE__struct *cfs, BYTE *dst) {
        BYTE tmp = 0;
        int count = 0;

        do {
                count++;
                cfs->CAB_fgetc(cfs, &tmp);
                if(dst) {
                        *dst  = tmp;
                        dst++;
                }

        } while(tmp);
        …
        Return count;
}
```

The above code is not good because it copies until a user controllable value is reached, regardless of the destination's size…like strcpy().