Panda Antivirus Library RemØte Heap Overflow
Security Advisory

Date
November 28, 2005

Vulnerability
The Panda Antivirus Library provides file format support for virus analysis. During decompression of ZOO files Panda 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.

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

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

http://www.pandasoftware.com/

Note: this library is also licensed to other venders with implementations that are likely affected, refer to Panda for specifics.

Credit
This vulnerability was discovered and researched by Alex Wheeler.

Contact
security@remØte.com

Panda Antivirus Library RemØte Heap Overflow Security Advisory

Description

The vulnerable code is responsible for Lempel-Ziv decompression of ZOO archive formats. Specifically, the vulnerability is the result of a loop constraint bounded by user input. An attacker may craft a section of codes to overwrite heap memory with arbitrary data. This technique can then be used to obtain complete control of the process importing the vulnerable code.

The following vulnerable code is from the pskcmp.dll file:


```
.text:010E0580                  mov     ecx, [esi+18Ch]
.text:010E0586                  mov     edx, [ecx+eax*4]
.text:010E0589                  mov     [esi+1A0h], edx
.text:010E058F
.text:010E058F loc_10E058F:                             ; CODE XREF: lempziv_decomp+24C↓j
.text:010E058F                  cmp     dword ptr [esi+1A0h], 0FFh
.text:010E0599                  jbe     loc_10E0809
.text:010E059F
.text:010E059F loc_10E059F:                             ; CODE XREF: lempziv_decomp+499↓j
.text:010E059F                  mov     edx, [esi+198h] ; ptr to current offset (194 initially)
.text:010E059F                                          ; into 3e94 byte buff
.text:010E05A5                  mov     ecx, [esi+190h] ; user controlled src table, 200A byte buff
.text:010E05AB                  dec     edx
.text:010E05AC                  mov     [esi+198h], edx
.text:010E05B2                  mov     eax, edx
.text:010E05B4                  mov     edx, [esi+1A0h]
.text:010E05BA                  mov     cl, [ecx+edx]
.text:010E05BD                  mov     [eax], cl
.text:010E05BF                  mov     edx, [esi+1A0h] ; current code
.text:010E05C5                  mov     eax, [esi+18Ch] ; user controlled code table ptr, 8082 byte buff
.text:010E05CB                  mov     eax, [eax+edx*4]
.text:010E05CE                  cmp     eax, 0FFh
.text:010E05D3                  mov     [esi+1A0h], eax
.text:010E05D9                  ja      short loc_10E059F
.text:010E05DB                  jmp     loc_10E0809
.text:010E05E0 ; ---------------------------------------------------------------------------
.text:010E05E0
.text:010E05E0 loc_10E05E0:                             ; CODE XREF: lempziv_decomp+416↓j
.text:010E05E0                  mov     eax, [esi+1A4h] ; current offset, 4108 byte buff
.text:010E05E6                  mov     ecx, [esi+7Ch]  ; end buff ptr, offset 1FEE into 4108 byte buff
.text:010E05E9                  cmp     eax, ecx
.text:010E05EB                  ja      loc_10E0809
.text:010E05F1                  mov     cl, [esi+1A0h]
.text:010E05F7                  mov     [eax], cl
.text:010E05F9                  mov     eax, [esi+1A4h]
.text:010E05FF                  mov     edx, [esi+198h]
.text:010E0605                  inc     eax
.text:010E0606                  mov     [esi+1A4h], eax
.text:010E060C                  mov     cl, [edx]
.text:010E060E                  mov     [eax], cl
.text:010E0610                  mov     edx, [esi+1A4h]
.text:010E0616                  mov     ecx, [esi+198h]
```

The above loop copies a byte based on an index into the source and code tables. The code and source tables are both controlled by the user. The boundary constraint only checks the next code value for over a byte to terminate the loop, which is not good. A special code sequence can be created to cause this loop to copy more than is intended. The simplest case is where a code indexes itself; a more useful case is when a sequence of codes indexes itself.

In order to exploit this vulnerability to execute arbitrary code three conditions must be met:

- The sequence of codes all must have values over a byte;
- At least one code or sequence must index itself; and,
- The code should be overwritten with a value to terminate the loop before the heap is destroyed.