

HISPASEC SISTEMAS

SEGURIDAD Y TECNOLOGÍAS
DE LA INFORMACIÓN

White paper:
**Microsoft Windows Vista animated mouse pointer
loader vulnerability analysis**

April 2007

1. Abstract

Microsoft Windows Vista, like previous versions, supports animated mouse pointer. The animated mouse pointers are loaded from .ANI files by the functions from USER32.DLL called LoadCursorIconFromResource and LoadCursor. These functions use an internal function called `_LoadCursorIconFromFileMap` which, under certain conditions, is prone to a buffer overflow.

2. Vulnerability description

To understand the vulnerability, a basic knowledge about the .ANI file format is required.

The .ANI file starts with a magic value “RIFF”, followed by a DWORD containing the data length, and followed by another magic value “ACON”. The rest of the file is based on chunks. Each chunk starts with a four byte identification word (“anih” for ANI header, “fram” for frames, etc), and a DWORD containing the chunk length. For example, the length of the “anih” ANI header is always 36 bytes long, so the eight bytes of the chunk would be “anih\x24\0\0\0”.

The internal function responsible for reading the animated pointer from the file is called `_LoadCursorIconFromFileMap` and is located in the USER32.DLL. The function tries to determine if the file has more than one frame, and if so, it calls another internal function called `_LoadAniIcon`. The pseudo code for the function follows:

```
riff_magic = Read 4 bytes ; read magic
if riff_magic != "RIFF"
return
Skip 4 bytes ; skip length
acon_magic = Read 4 bytes ; read ACON magic
if acon_magic != "ACON"
return
tag = Read 8 bytes ; tag is a struct (magic, size)
if tag.magic == "anih"
if tag.size != 0x24 ; size check
return
header = Read tag.size bytes ; read the chunk
result = ValidateAnih(header) ; validate the header
if !result
return
if header.cFrames > 1 ; are there two or more frames?
call _LoadAniIcon ; execute the LoadAniIcon function
return
...
```

The size of the header is verified in this function in two places, first the “if tag.size != 0x24”, and then by the ValidateAnih function. If there is more than one frame, an internal function _LoadAniIcon is called. The function pseudo code follows:

```

Rewind ; rewinds the file
Skip 8 bytes
acon_magic = Read 4 bytes ; read the ACON magic once again
if acon_magic != "ACON"
return
loop:
tag = Read 8 bytes ; tag is a struct (magic, size)
if tag.magic == "seq "
...
else if tag.magic == "LIST"
...
else if tag.magic == "rate"
...
else if tag.magic == "anih" ; is this the header ?
header = Read tag.size bytes ; read the header
result = ValidateAnih(header) ; validate the header
if !result
return
...
else
Skip tag.size bytes
goto loop
  
```

The chunks are read in a loop sequentially, and handled if supported. The problem lies on the line where the “anih” magic is found, and the header is read. Since the size is not validated, it could be incorrectly supplied causing data on the stack to be overwritten. Please note that the size of the first “anih” header is validated in the parent function, but since there is no mechanism checking if the “anih” chunk was already supplied, it may be supplied more than one time, and the second instance size will not be verified before the reading. The ValidateAnih function, called after the reading, would determine the header as invalid, since the size would differ from 0x24 bytes, but that is just too late.

3. Malware analysis

A sample live malware I analyzed, that used this vulnerability was Trojan-Downloader.Win32.Ani.g . After a successful buffer overflow, it executes the LoadLibraryA to load URLMON.DLL library, then it downloads a specified executable file (using URLDownloadToCacheFileA function), and executes it using CreateProcessA(“cmd”, “/c <name>”,...). The offsets of the used functions are not hardcoded, but rather than that, the malware has a built-in GetProcAddress-like function that uses a simple hash algorithm to locate the right function. For example, the URLDownloadToCacheFileA function is acquired using the following code:

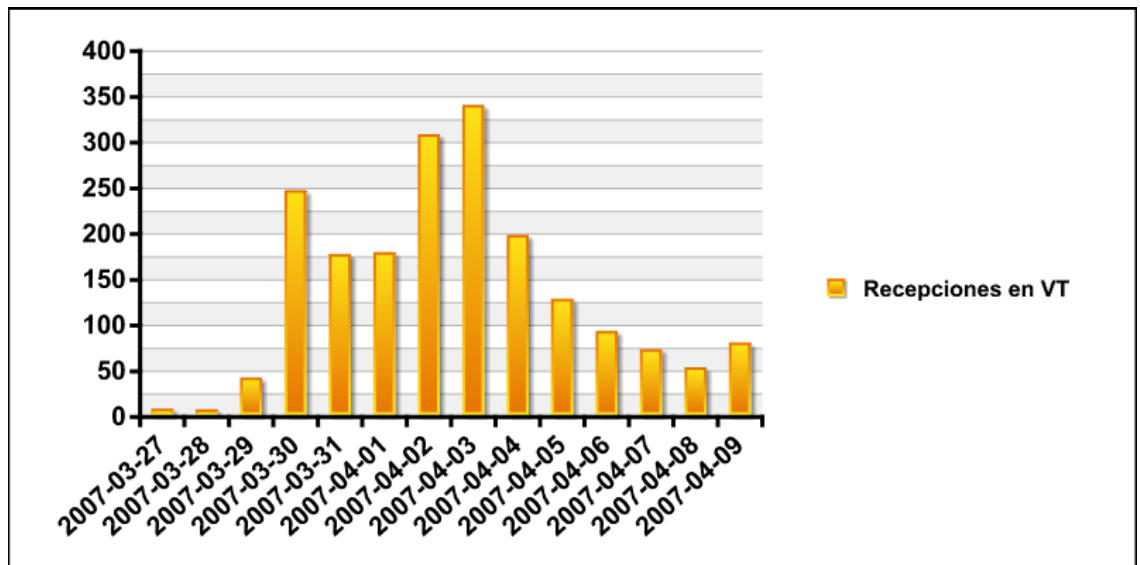
```
call sub_1A0 ; this acquires the handle to URLMON.DLL
push 54FEF4Fh ; the hash of the sought function
push eax ; the handle to URLMON.DLL
call get_func_by_hash ; the right call
```

```
get_func_by_hash:
...
try_next_func:
...
calc_hash: ; this loop calculates the hash
lodsb
test al,al
jz hash_done
ror edi, 0x0D ; the heart of the hash
add edi, eax ;
jmp calc_hash
cmp edi, [arg_sought_hash]
jnz try_next_func
...
```

The usage of such functions makes it a little more system version independent.

4. VirusTotal Reception

This graphs shows the number of samples that exploits ANI vulnerability received in VirusTotal from 27-03-2007 to 09-04-2007. Total number of unique samples received during this period is 1037 (based on MD5 hashes).



5. Summary

Since it is very simple to create such an exploit, it is recommended to update the Microsoft Windows with a patch fixing this issue.

The analysis was made on Microsoft Windows Vista version 6.0.6000.16386.