KAIST CSRC Cyber Security Research Center 보안분석보고서

Document #	CSRC-12-03-006	Title	XMLCoreServices Vulnerability Analysis (CVE-2012-1889)		
Type	☐ Attack Trend ☐ Technical Analysis ■ Specialty Analysis	Date	July 6, 2012	Author	KAIST Graduate School of Information Security Minsu Kim

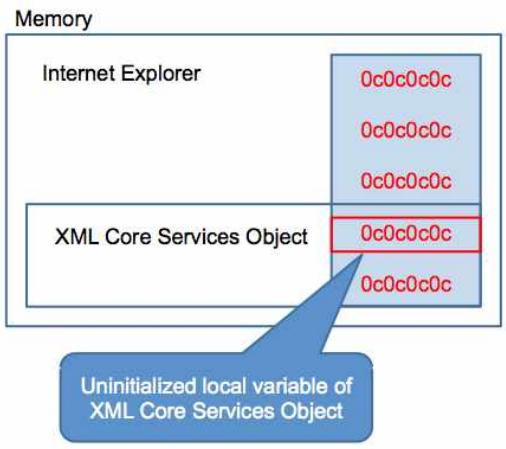
* Keyword : XMLCoreServices, CVE-2012-1889

1. Executive Summary

Recently, the malicious web pages exploiting XMLCoreServices vulnerability are frequently observed, and since Microsoft have released just a temporary fix for this vulnerability, many Internet Explorer users are exposed to this security threat. This document provides detailed analysis of XMLCoreServices (CVE-2012-1889) vulnerability.

This vulnerability can be exploited by abusing uninitialized memory section of Microsoft Core Services 3.0, 4.0, 5.0 and 6.0, and ultimately executes malicious code injected by the attacker. This vulnerability can be temporarily removed by Fix It (http://support.microsoft.com/kb/2719615), which disables XML Core Services, however Microsoft should release official patch to this vulnerability as soon as possible.

This vulnerability has been analyzed on the machine with Windows XP SP2, Internet Explorer 6, and Microsoft Core Services 3.0. The vulnerability exists in msxml3.dll, which provides Core Services. The structure of memory where the exploitation of the vulnerability takes place is shown in Figure 1 below.



[Figure 1] Memory structure upon the exploit

2. Description

1. The Exploit Code

Figure 2 is a part of the source code that exploits XMLCoreServices vulnerability written in JavaScript language.

```
var gondad = document.createElement('object');
gondad.setAttribute("classid","clsid:f6D90f11-9c73-11d3-b32e-00C04f990bb4");
gondad.setAttribute("id","oo");
document.body.appendChild(gondad);
var shellcode = unescape('%u9090%u9090생략%uBDBD%uC3C3');
var a = new Array();
var ls = 0x100000-(shellcode.length*2+0x01020);
var b = unescape("%u0d0d%u0d0d");
                                        .
                                        .
                                    생 략
                                        .
                                        .
var obj = document.getElementById('oo').object;
var src = unescape("%u0c0c%u0c0c");
while (src.length < 0x1002) src += src;
src = "\\\\xxx" + src;
src = src.substr(0, 0x1000 - 10);
var pic = document.createElement("img");
pic.src = src;
pic.nameProp;
obj.definition(1);
```

[Figure 2] Source Code of Malicious Web Page

The heap section of Internet Explorer has to be modified in order to place the shellcode on the desired memory location, however this document does not precisely describe how it is done. For more information, refer to Heap Feng Shui in JavaScript[1].

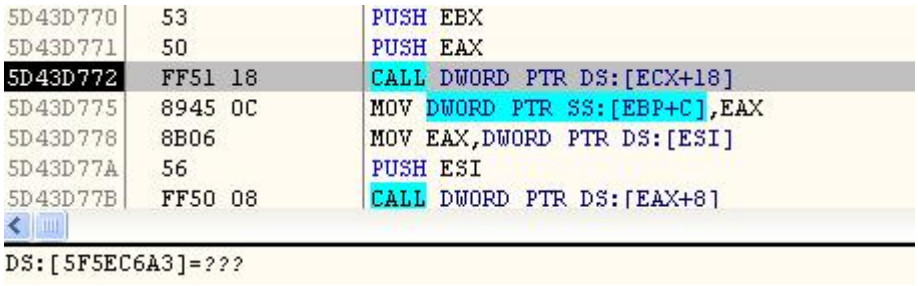
The heap section is consisted of two parts. One is nop-sled, and the other is shellcode. This malicious shellcode section is followed by the nop-sled section containing meaningless code(0x0D0D0D0D) thus only increases EIP address to eventually execute the shellcode. This typical method is known as Heap Spray.

As shown in Figure 2, it creates an element called 'img' in the object of the webpage, and it assigns variable called 'src' containing 0x0C0C0C0C to 'img'. By abusing 'img' element, it will write 0x0C0C0C0C in the location where XML Core Services' definition() function refers to, and ultimately, the written location will be executed.

2. Vulnerability Analysis

A. Vulnerable Spot

In order to figure out where the vulnerable spot is, we temporarily removed the shellcode from the malicious page, and attached the debugger to this page. As shown in Figure 3, access violation occurs at 0x5D43D772, since the shellcode has been removed. The corresponding section belongs to msxml3.dll, specifically _dispatchImpl::InvokeHelper function.



[Figure 3] Vulnerable Spot

B. Flow Analysis

From the previous section, we have figured out that the vulnerability gets triggered by the function called _dispatchImpl::InvokeHelper. Since this function is responsible for being exploitable, we have put breakpoint to this function for the analysis.

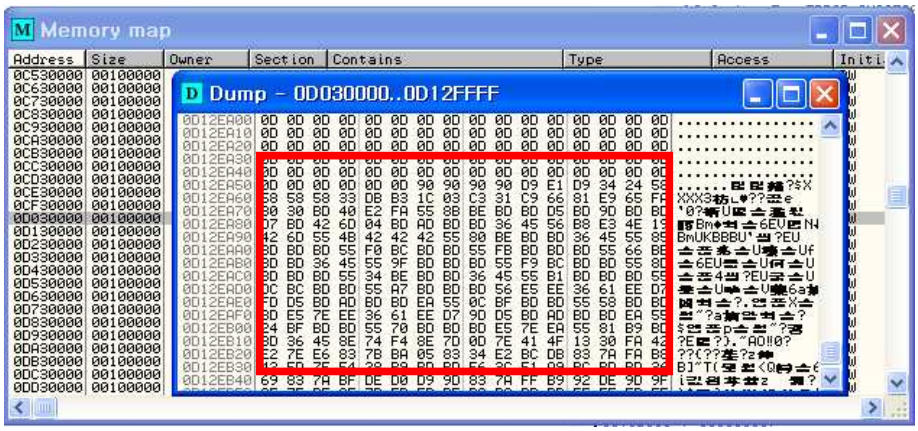
This function gets called three times in total, and we learned that the third one actually exploits the vulnerability. Each call and the corresponding web page source code is described in Table 1. The second call and the third call were doing important jobs, hence we explain them in this document. We will begin with the analysis of the third call for better understanding.

1st Call	gondad.setAttribute("classid","clsid:f6D90f11-9c73-11d3-b32e-00C04f990bb4");
2nd Call	gondad.setAttribute("id","oo");
3rd Call	obj.definition(1);

[Table 1] InvokeHelper Calls

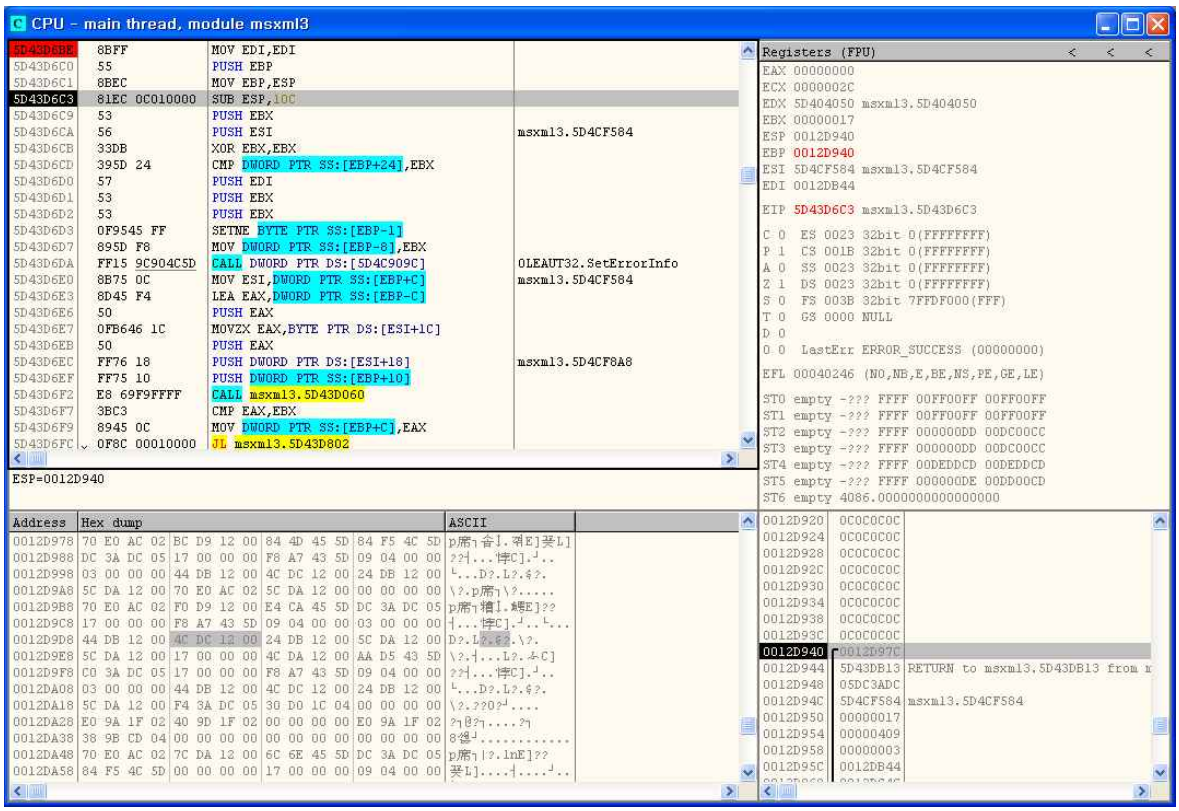
C. The Third Call of InvokeHelper

In Figure 4, the memory dump of the Internet Explorer process right before the third call of InvokeHelper function shown, and it is sprayed with 0x100000 sized heap.



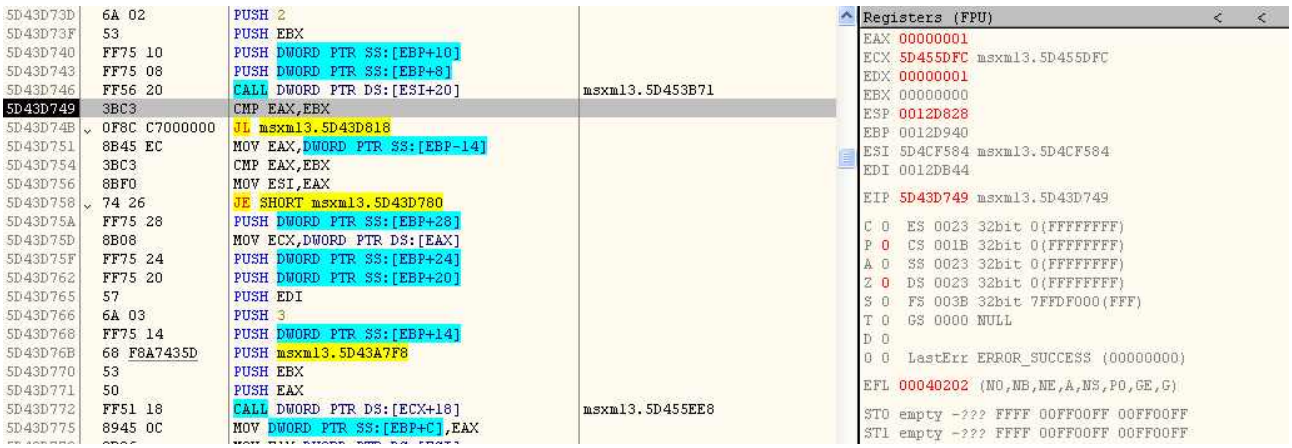
[Figure 4] Heap Spary

The red boxed section of Figure 4 is the shellcode, which is followed by the nop-sled(0x0D0D0D0D) to aid its execution. Knowing that the shellcode is already loaded in the memory before the third call, we need to inspect how the third call will invoke the execution of the shellcode. The beginning of the code section of InvokeHelper function in msxml3 module is shown in Figure 5.



[Figure 5] The beginning section of msxml3.dll's InvokeHelper function

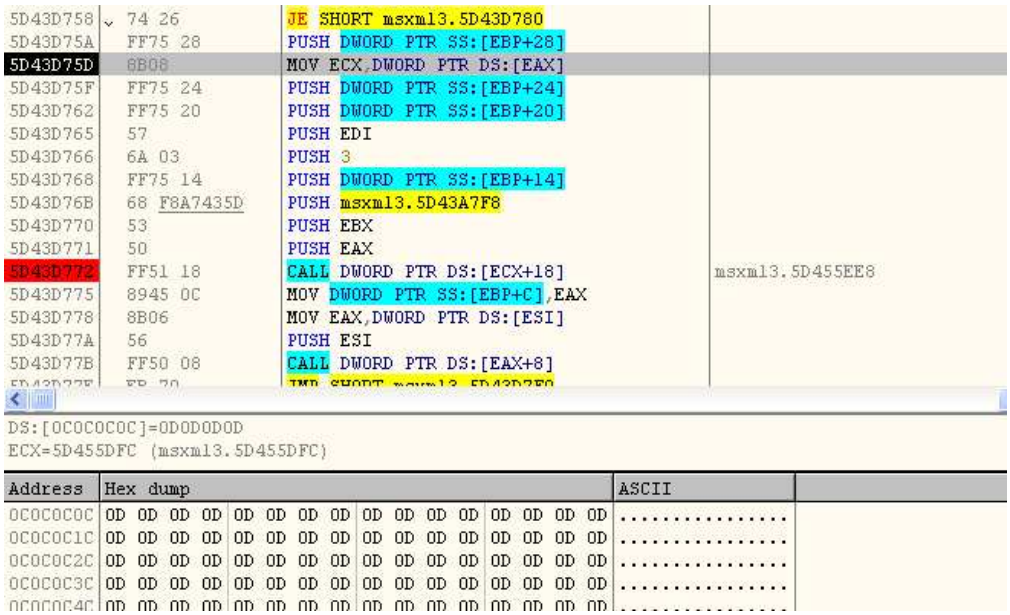
By carefully executing each line of the code in Figure 5, we can reach the point right before the execution of the command at 0xD43D6C3(SUB ESP, 10C). To be specific, we stopped executing the program right before it reserves 10C bytes for the local variables of this function. At this moment, the stack pointer at this moment points to 0x0012D940, and the space reserved for local variables of this function is already sprayed with 0x0C0C0C0C. In other words, by referring to uninitialized local variable in this function to call something else, attacker can handle the EIP register in order to execute the shellcode that is already loaded on the memory. The reason that the shellcode can only be executed by definition() function is shown in Figure 6.



[Figure 6] Branch to vulnerable code execution

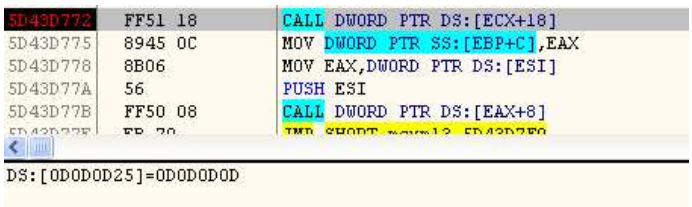
The command CALL DWORD PTR DS:[ESI+20] at 0x5D43D746 executes the command at 0x5D453B71 of msxml3.dll, which is DOMNode::_invokeDOMNode function. Node::getDefinition sets EAX value to 1, and therefore 0x5D43D74B of Figure 6 does not branch and the vulnerable code can be executed.

The vulnerable spot was 0x5D43D772 in Figure 6, and at this point, the program calls the value that ECX+18 points to. This ECX register gets modified at 0x5D43D75D in Figure 6, by storing the value of the memory pointed by EAX register. This EAX register stores the value of EBP-14 at 0x5D43D751. Since EBP is 0x0012D940, EAX stores the value at 0x0012D92C. Since the variable is not initialized upon the execution of this code, EAX gets 0x0C0C0C0C value. The memory structure at 0x5D43D75D within the debugger is shown Figure 7.



[Figure 7] Memory dump at 0x5D43D75D

As shown in Figure 7, EAX contains 0x0C0C0C0C and it points to the value of 0x0D0D0D0D, hence this 0x0D0D0D0D value will be stored in ECX upon the execution of 0x5D43D75D.

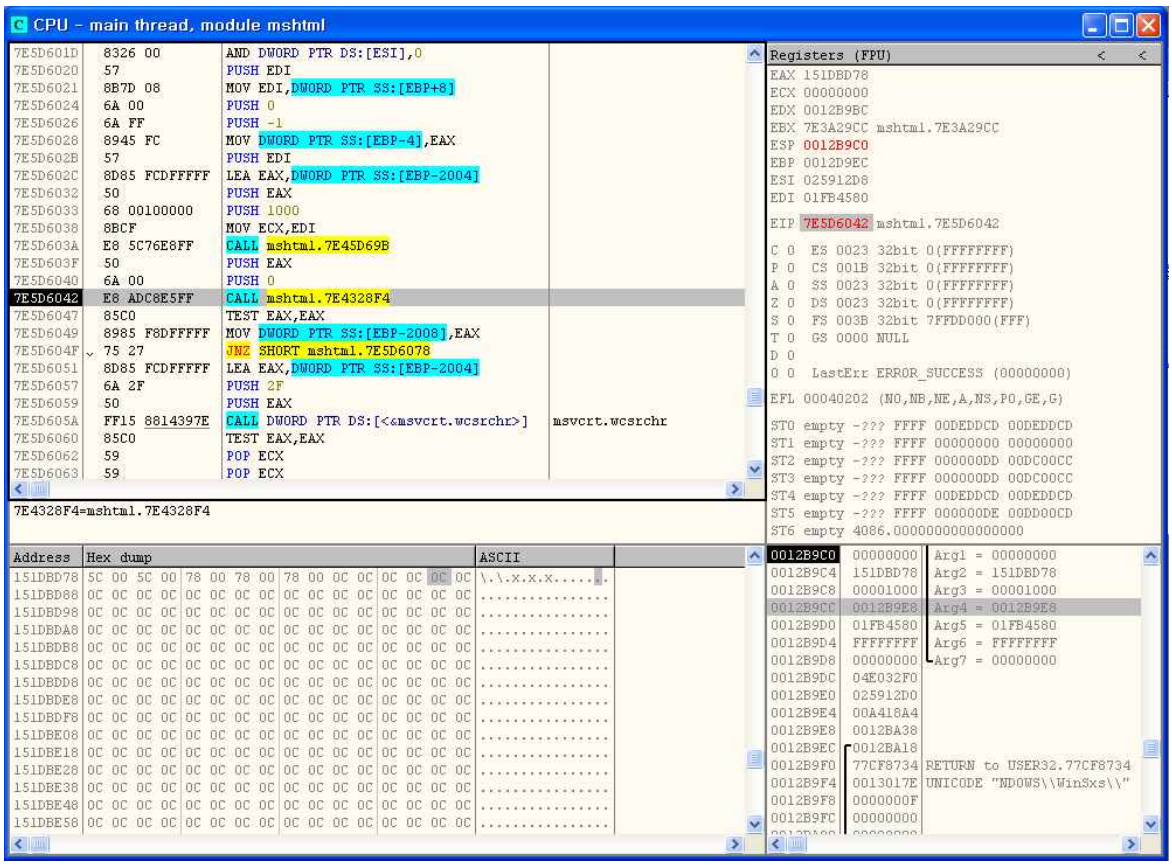


[Figure 8] Memory dump at 0x5D43D772

Eventually, as shown in Figure 8, the command at 0x0D0D0D0D, which the attacker placed in the memory, gets executed as we expected.

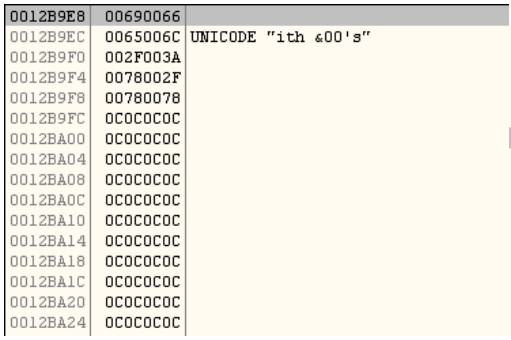
D. The second call ~ the third call

Before the third call, the value of 0x0C0C0C0C was assigned to the local variable area(0x0012D93C) of msxml3.dll's InvokeHelper. We need to trace where this assignment is made. We have placed the hardware breakpoint on this memory location of 0x0012d93C in order to be notified upon the access. During the trace, the command shown in Figure 9 was discovered.

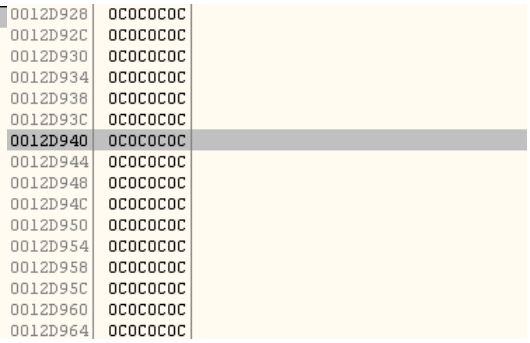


[Figure 9] pic.src = src section of code

The command calls the function that takes seven parameters at 0x7E5D6043 in Internet Explorer's mshtml module. This function is called when pic.src = src code shown in Figure 2 gets executed, and this code assigns 0x0C0C0C0C value to 'src' property of 'img' object of the webpage. The address of the second parameter of this function is 0x151DBD78, and this location contains the value of 'src'. The fourth parameter is the relative address of 'pic.src', and copies the value of 'src' to this 0x0012B9E8 memory location.



[Figure 10] Memory Dump after The Execution

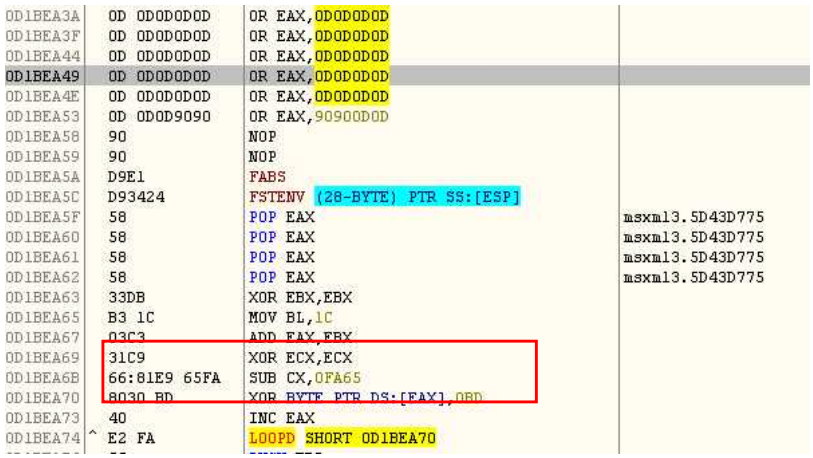


[Figure 11] Memory Dump after The Execution

Figure 10 and 11 shows the state of memory after the execution of the command at 0x7E5D6042. Figure 10 shows where the value is assigned initially, and Figure 11 shows the memory section where vulnerable spot refers to. In both Figure 10 and 11, the memory is sprayed with 0x0C0C0C0C after the execution. Therefore, attacker can successfully exploit this vulnerability as desired.

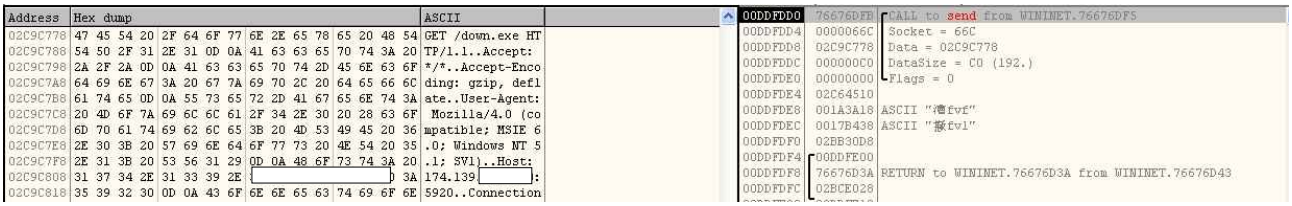
E. After the exploit

Once the exploit code is executed, the program moves on to 0x0D0D0D0D section, which is sprayed by the attacker. Beginning from 0x0D0D0D0D section, it bumps into the nop-sled that gently leads to the shellcode section, as shown in Figure 12.



[Figure 12] shellcode and decoding routine

The section in the red box in Figure 12 designates the decoding routine that decodes the obfuscated string values, and after this routine, the execution of the attacker's shellcode begins.



[Figure 13] Malware download section

As shown in Figure 13, the shellcode makes GET request to create malware from 174.139.XXX.XXX/down.exe and executes it. Since the shellcode serves as a downloader, this additionally downloaded file, such as down.exe, can perform any malicious activity as desired. The most frequently observed activities of the downloaded programs are killing anti-virus programs, hijacking accounts and installing backdoors.

3. Conclusion

XML Core Services (CVE-2012-1889) vulnerability is exploitable by abusing uninitialized variable in its object. For this to work, locations of 'img' obejct and XML Services object should be well-controlled by calling CollectGarbage function and assigning values to heap section, however the analysis of this specific part has been omitted in this document.

One of possible scenarios of the attack exploiting this vulnerability begins with an attacker modifying the normal webpage by attacking vulnerable website, and thus directing the visitors to the malicious webpage. This malicious webpage declares vulnerable XML Core Services object and assigns it to the DOM object of the webpage. If XML Core Services is disabled, the exploitation will fail to take place. The webpage then assigns malicious shellcode to the heap section of the Internet Explorer process, and writes the address of the heap section repeatedly in 'src' property of 'img' object. By overwriting on the section of the local variable that will be called upon the execution of XML Core Services' definition function, the shellcode in the heap section can be executed by the visitor's machine. This embedded shellcode usually acts as the downloader that downloads and executes additional malware, and therefore, the visitors can be infected by simply visiting this malicious webpage. The common malicious activities are hijacking personal informations, such as game accounts and etc., or installing backdoors for additional attacks.

Currently, all of the Internet Explorer users are exposed to such threat, since msxmlx.dll that provide XML Core Services is loaded upon the execution of Internet Explorer as default library. Especially, if a popular website contained such malicious page, most of the visiting machines without the temporary patch would be infected.

Microsoft should release the official patch to remove this vulnerability as soon as possible, because the attack is expected to be highly successful without official patch.

3. References

1. Sotirov. A, Heap feng shui in JavaScript, Black Hat Europe, 2007.
2. http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-1889
3. http://technet.microsoft.com/ko-kr/security/advisory/2719615
4. Brian MARIANI & Frédéric BOURA, https://www.htbridge.com/publication/CVE-2012-1889.pdf