KAIST CSRC Cyber Security Research Center 보안분석보고서

문서번호	CSRC-12-03-006	제 목	XMLCoreServices 취약점 분석 (CVE-2012-1889)		
종 류	□ 공격동향 □ 기술분석 ■ 전문분석	작 성 일	2012년 7월 6일	작 성 자	KAIST정보보호대학원 김민수

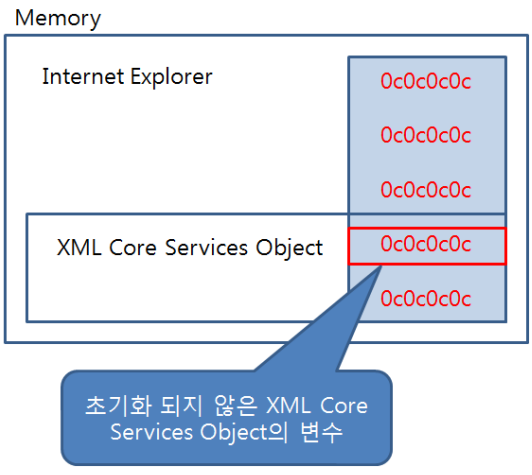
* Keyword : XMLCoreServices, CVE-2012-1889

1. 요 약 Executive Summary

최근 XMLCoreServices 취약점으로 인한 악성코드 유포사례가 늘고 있고, 아직 임시패치밖에 나오지 않은 상황이라 많은 사용자들이 인터넷 이용 시 이 취약점을 통한 악성코드 감염위협에 노출되어 있다. 본 분석 보고서에서는 XMLCoreServices (CVE-2012-1889) 취약점에 대한 상세한 분석을 제공하고자 한다.

해당 취약점은 Microsoft XML Core Services 3.0, 4.0, 5.0, 6.0에서 초기화 되지 않은 메모리 부분을 이용함으로써, 공격자의 악의적인 코드가 실행될 수 있는 취약점이다. 해당 취약점은 http://support.microsoft.com/kb/2719615 에서 fix it 으로 해당 서비스를 비활성화 시킴으로써 임시적으로 취약점을 제거할 수 있으나 시급하게 정식패치가 요구된다.

본 분석은 Windows XP SP2, Internet Explorer6, Microsoft Core Services 3.0환경에서 진행되었다. 취약점이 발생하는 부분은 Core Services를 담당하고 있는 msxml3.dll에서 발생하였다. 취약점이 발생하는 개략적인 원인은 [그림 1]과 같다.



[그림 1] 취약점 발생 시 메모리 구조

2. 설 명 Description

1. Exploit Web Page

해당 취약점을 발생시키는 web page의 코드 중 일부는 [그림 2]와 같다.

```
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);
```

[그림 2] 공격 페이지 코드

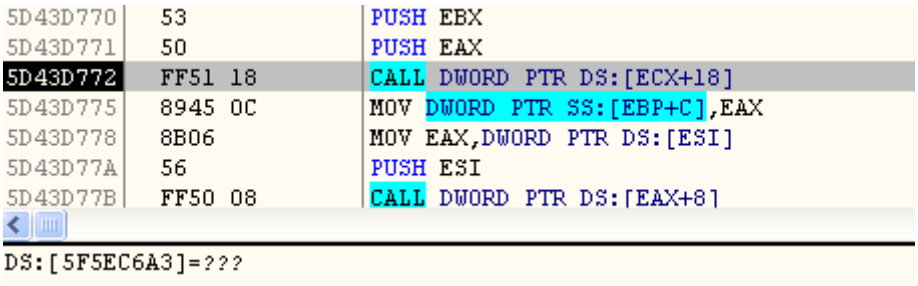
원하는 메모리 위치에 Shellcode를 위치시키기 위해 IE Heap영역을 조절하는 부분은 본 보고서에는 생략하고자 한다. 해당 부분은 Heap Feng Shui in JavaScript[1]를 참조하기 바란다. IE의 힙 영역에는 0x0D0D0D0D의 Nopsled역할(의미 없는 명령어로 단지 EIP레지스터의 주소만 증가시킴)을 하는 부분과 공격자가 작성한 악성 Shellcode가 할당되어 있다. 전형적인 HeapSpray기법을 이용하고 있다.

웹 페이지의 객체에 img객체를 만들고 여기의 src에 0x0C0C0C0C 값을 할당 시킨다. 전체적인 코드 흐름을 보면 Nopsled+Shellcode를 Internet Explorer의 힙영역에 뿌려 놓고, XML Core Services 객체의 definition함수가 참조할 변수를 img객체를 이용하여 0x0C0C0C0C로 덮어 써 코드 흐름이 0x0C0C0C0C주소로 이동하게 만든다.

2. 취약점 분석

가. 취약점 발생 지점

어떤 부분에서 취약점이 발생하는지 알아보기 위해 쉘 코드를 제거하고 웹 페이지를 실행시켜 보았다. [그림 3]과 같이 0x5D43D772부분에서 쉘코드를 제거하였기 때문에 Access Violation이 발생하였다. 해당 부분은 msxml3.dll의 코드 영역이며 더 정확히 _dispatchImpl::InvokeHelper함수의 코드 영역이다.



[그림 3] 취약점 발생 지점

나. 흐름 분석

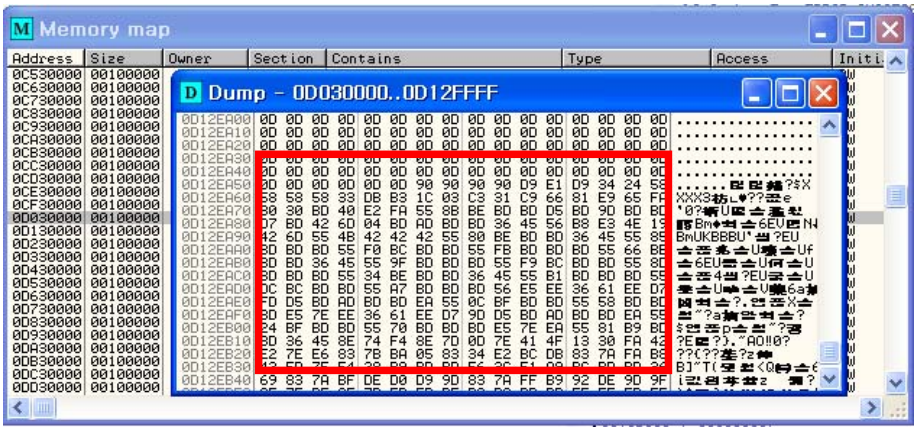
취약점이 발생하는 함수는 _dispatchImpl::InvokeHelper함수에서 발생한다. 해당 함수에 브레이크 포인트를 걸고 디버깅을 시작했다. 총 해당함수가 세 번 호출되는데 3번째 호출될 때 취약점이 발생하는 코드가 실행된다. 해당 함수 호출 지점과 코드와 연관시키면 [표 1]과 같다. 중요한 호출 지점이 2번째 호출지점 이후와 3번째 호출지점인데 두 부분으로 나누어 분석하도록 하겠다. 이해를 돕기 위해 3번째 호출지점 이후부터 분석을 진행하고자 한다.

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

[표 1] InvokeHelper함수 호출 지점

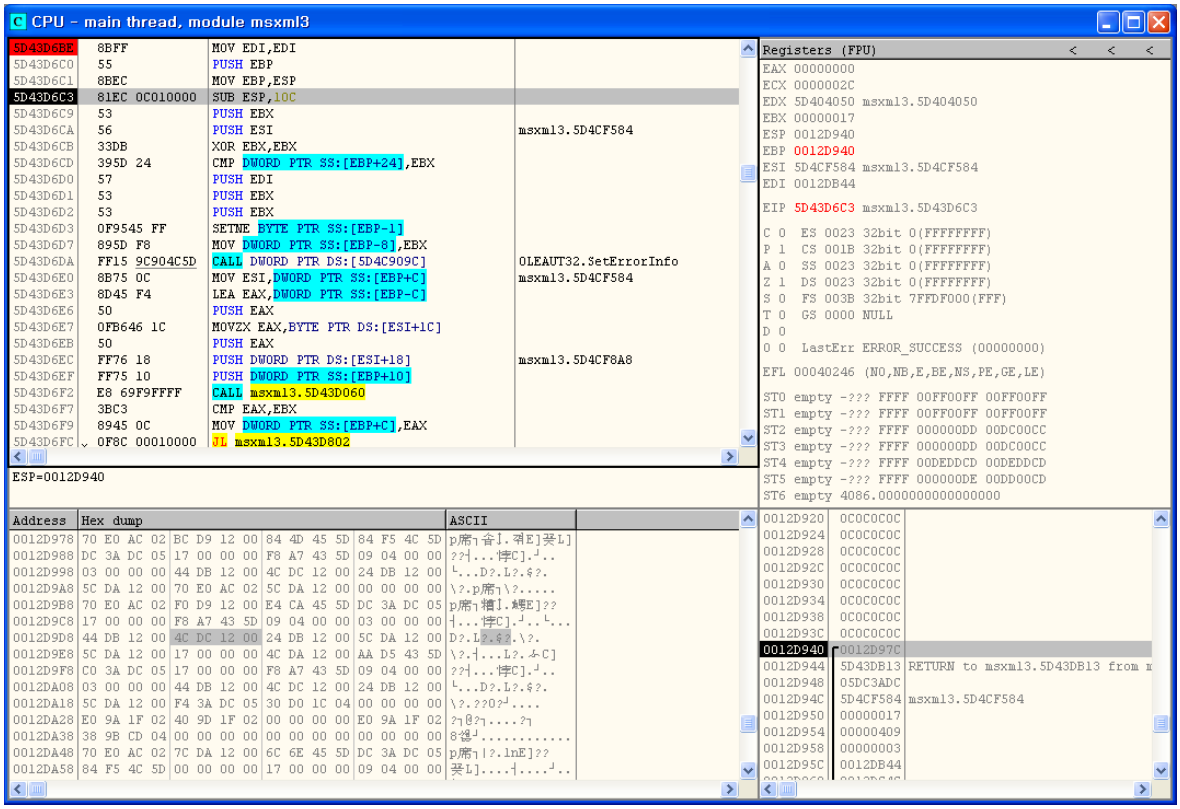
다. 3번째 호출지점

3번째 InvokeHelper함수가 실행되기 직전의 Memory상태를 상태를 살펴보면 [그림 4]와 같이 Internet Explorer의 Memory에 0x100000크기의 Heap이 Spray 되어 있는 모습을 볼 수 있다.



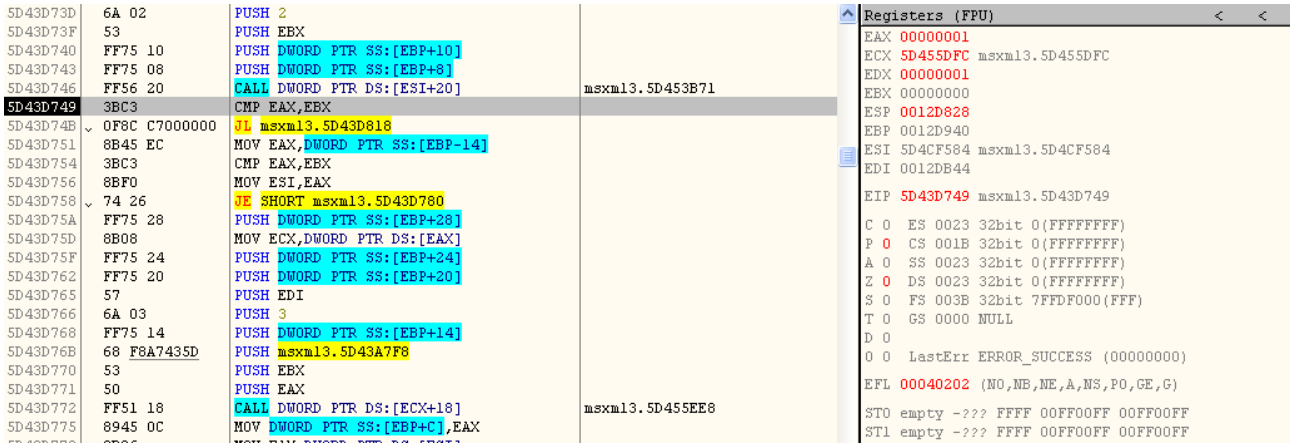
[그림 4] Heap Spary

빨간색 테두리 부분의 9090값이 있는 지점부터 공격자의 Shellcode부분이며 그 앞에는 공격코드가 잘 실행될 수 있도록 하는 0x0D0D0D0D NopSled부분이 붙어 있는 것을 확인 할 수 있다. Shellcode가 이미 메모리에 있는 것을 확인하였고, 이제 어떻게 3번째 InvokeHelper함수에서 Shellcode가 할당된 코드 부분으로 흐름이 이동되는지 살펴보도록 하겠다. msxml3.dll의 InvokeHelper의 함수 시작부분 코드는 [그림 5]와 같다.



[그림 5] msxml3.dll의 InvokeHelper함수 시작부분

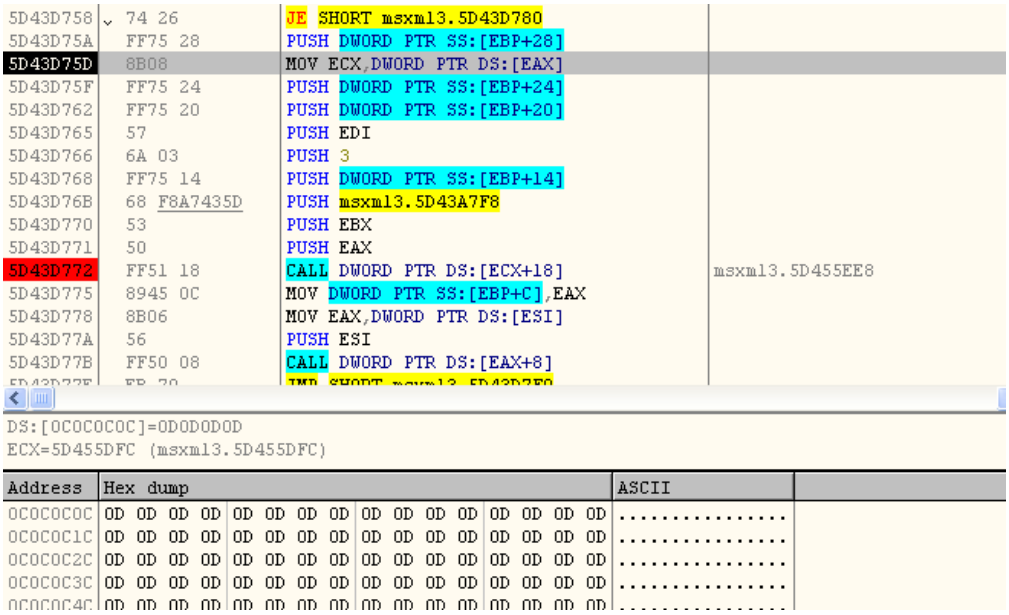
[그림 5]의 코드를 한 줄씩 실행시켜 0x5D43D6C3 명령어를 실행시키기 전의 스택 상태를 살펴보면 스택포인터는 0x0012D940을 가지고 있다. 그러나 해당 함수의 지역변수를 위한 공간을 할당하는 SUB ESP, 10C 명령어를 실행시키기 전에 해당 지역변수를 할당 할 공간에 이미 0x0C0C0C0C값이 들어가 있는 것을 확인 할 수 있다. 만약 이 함수의 초기화 시키지 않은 지역변수를 참조하여 다른 함수를 호출하는 코드가 있다면 공격자가 이미 메모리에 할당한 쉘코드를 실행시킬 수 있다. InvokeHelper함수가 3번이나 호출되면서 실제로 취약점이 일어나는 지점의 코드실행이 definition함수를 호출 할 때만 일어나는 이유는 [그림 6]을 통해 확인 할 수 있다.



[그림 6] 취약점이 발생하는 코드를 실행시킬 수 있는 분기 지점

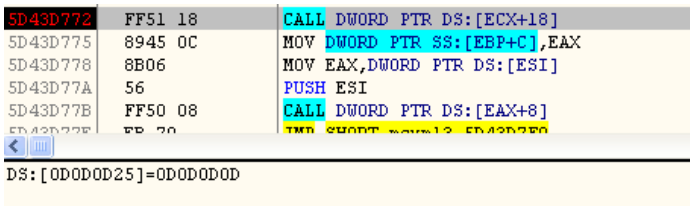
0x5D43D746에 있는 명령어인 CALL DWORD PTR DS:[ESI+20] 지점은 msxml3.dll의 0x5D453B71주소에 있는 명령어를 실행시키는데 해당 주소에 있는 함수는 DOMNode::_invokeDOMNode 함수이다. 자세한 분석은 생략하고 최종 호출 하는 함수는 Node::getDefinition함수가 EAX값을 1로 세팅하여 [그림 6]의 0x5D43D74B지점에서 분기하지 않고 취약점이 발생하는 명령어가 실행 될 수 있다.

실제로 취약점이 발생했던 위치가 [그림 6]의 0x5D43D772 지점이었으며 이 때 ECX레지스터 값+18 메모리 위치에 있는 값을 참조하여 호출한다. 이 ECX 레지스터의 값을 변경시키는 부분을 보면 [그림 6]의 0x5D43D75D에서 변경이 된다. 여기서는 ECX 레지스터에 EAX 레지스터가 가지고 있는 값의 메모리 위치에 있는 값을 저장한다. 이 EAX레지스터의 값을 변경시키는 부분을 찾기 위해 다시 코드를 거슬러 올라가면 [그림 6]의 0x5D43D751에서 변경이 되며 이 때 EAX 레지스터에는 EBP-14에 있는 값이 저장된다. EBP가 0x0012D940이므로 EAX에는 [0x0012D92C]에 있는 값이 저장되게 된다. 이 코드가 실행될 때 시점도 변수를 초기화 하지 않아 [그림 5]의 스택과 같은 상태라서 EAX 레지스터에는 0x0C0C0C0C가 들어가게 된다. 0x5D43D75D시점의 메모리 상태를 보면 [그림 7]과 같다.



[그림 7] 0x5D43D75D 시점의 메모리 상태

[그림 7]을 보면 위의 설명과 같이 EAX 레지스터에 0x0C0C0C0C값이 저장되어 있고 해당 메모리 주소를 살펴보면 0x0D0D0D0D가 저장되어 있음을 확인 할 수 있다. 0x5D43D75D명령어가 실행되고 나면 ECX에는 0x0D0D0D0D값이 저장되어 있다.

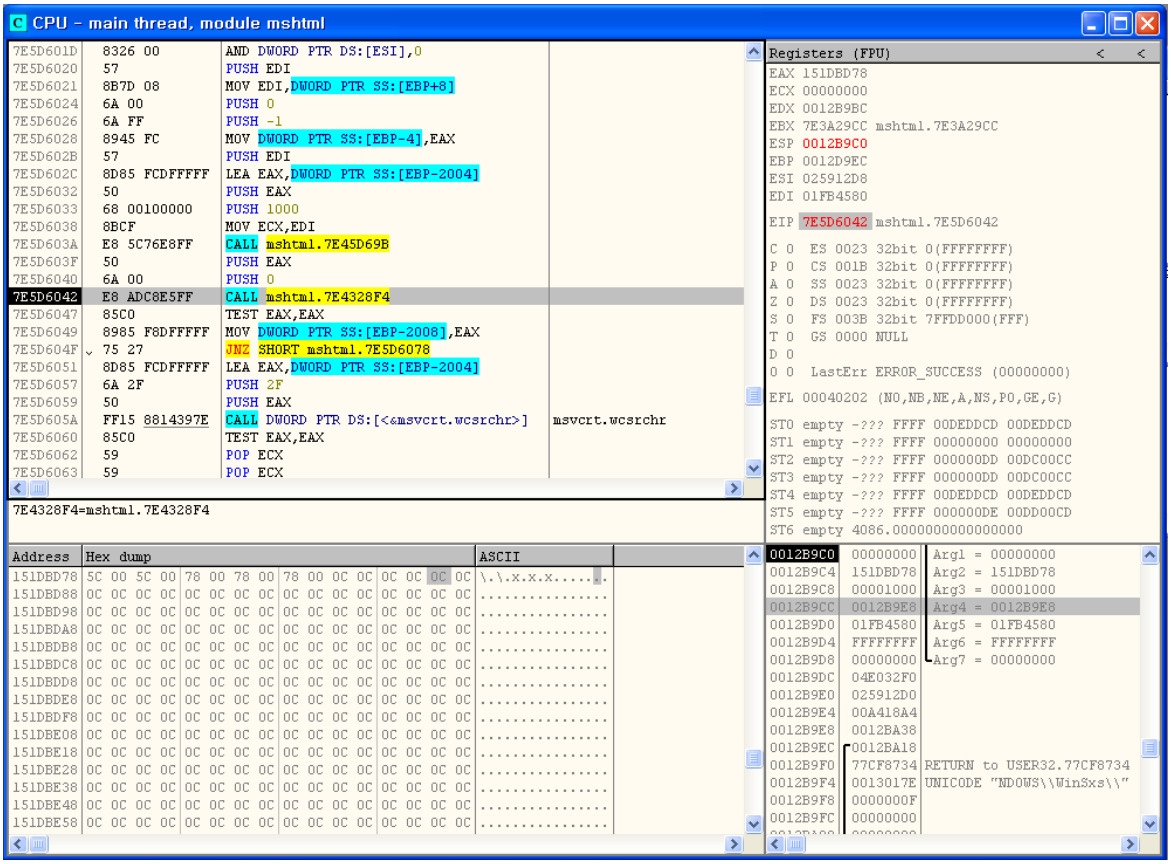


[그림 8] 0x5D43D772 시점의 메모리 상태

[그림 8]을 보면 본 보고서에서 예상 했던 대로 공격자가 위치시켜 놓은 0x0D0D0D0D 주소의 명령어를 실행하게 된다.

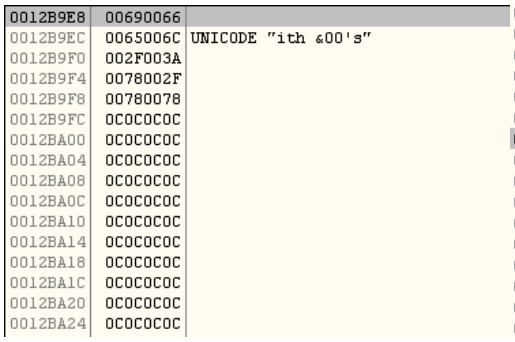
라. 2번째 호출지점 ~ 3번째 호출지점

3번째 obj.definition(1); 함수가 호출되기 전에 msxml3.dll의 InvokeHelper함수 시작지점에 이미 로컬 변수 영역에 0x0C0C0C0C값이 할당되어 있었다. 언제 할당되었는지 살펴보고자 한다. 이미 0x0012D93C 주소에 0x0C0C0C0C값이 저장되는 사실을 알고 있다. 이 사실을 바탕으로 해당 주소에 하드웨어 브레이크 포인트를 걸고 해당 주소를 변화시키는 함수를 추적하다보면 [그림 9]와 같은 명령어를 발견할 수 있다.

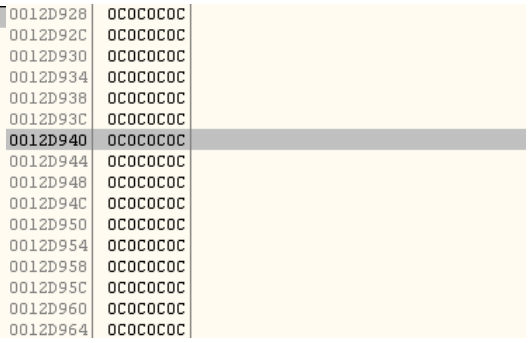


[그림 9] pic.src = src 부분의 코드

Internet Explorer의 mshtml모듈 0x7E5D6042지점에 7개의 인자를 받아 실행하는 함수가 있다. 해당 부분은 웹 페이지의 pic.src = src코드 실행 부분이며, 앞의 pic는 웹 페이지의 img객체이며 이 웹페이지의 img객체의 src속성에 0x0C0C0C0C를 반복한 값을 가지고 있는 src변수를 할당하는 부분이다. 두 번째 변수의 주소인 0x151DBD78 메모리 영역을 살펴보면 웹페이지에서 src 변수에 할당했던 문자열을 확인할 수 있다. 4번째 인자는 pic.src와 관련이 있는 주소값이며 이 0x0012B9E8 메모리 영역에 src변수에 있던 값을 복사한다.



[그림 10] 함수 실행 이후 메모리

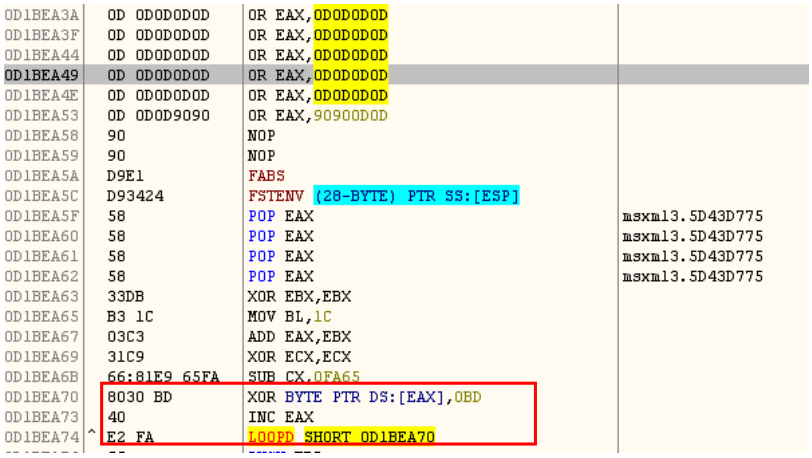


[그림 11] 함수 실행 이후 메모리

0x7E5D6042지점의 명령어가 실행 된 직후 메모리를 살펴보면 값이 처음 할당되는 시작지점인 [그림 10]과 나중에 취약점이 일어나는 부분이 참조하는 메모리 영역인 [그림 11]과 같다. 취약점이 일어났던 지점에서 참조하였던 0x0012D92C주소의 값도 0x0C0C0C0C로 바뀌어 있음을 확인 할 수 있다. 취약점이 있는 부분에서 이 주소부분의 값을 참조하기 때문에 공격자의 의도대로 공격을 성공시킬 수가 있다.

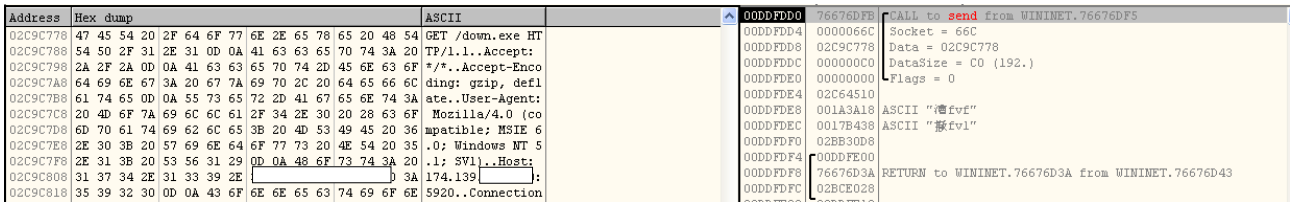
마. 취약점 발생 이후

취약점 발생 위치의 명령어가 실행되고 나면 공격자가 Heap에 뿌려 두었던 0x0D0D0D0D로 코드 흐름이 변경된다. 코드 흐름이 변경되고 나면 최초 NopSled를 만나게 되고, NopSled가 끝나고 나면 [그림 12]와 같이 공격자가 의도한 Shellcode지점을 볼 수 있다.



[그림 12] Shellcode 및 Shellcode 디코딩 루틴

[그림 12] 부분에 Shellcode 문자열을 숨기기 위하여 인코딩 한 코드를 복호화 하는 루틴이 있으며, 복호화 루틴이 끝나게 되면 본격적으로 공격자가 의도한 코드를 분석할 수 있다.



[그림 13] 악성코드 다운로드 부분

Shellcode분석을 진행하다가 보면 [그림 13] 과 같이 174.139.XXX.XXX/down.exe라는 악성코드 파일을 GET 요청을 하여 파일로 생성하여 실행하는 Downloader기능을 하는 Shellcode였다. 이 추가적으로 받아지는 down.exe파일은 어떠한 행위라도 할 수 있으며, 많이 발견되는 행위는 백신을 무력화 한 뒤 계정 탈취 및 백도어 등 다양한 행위를 수행할 수 있다.

3. 종합

XML Core Services (CVE-2012-1889) 취약점은 해당 객체에서 초기화 되지 않는 변수를 참조하여 발생하는 취약점이다. 해당 취약점을 발생시키기 위해, img객체와 XML Services객체를 할당하는 위치를 CollectGarbage함수와 Heap 영역 할당을 통해 조절해야 한다. 이 부분에 대한 분석은 생략하였다.

전체적인 공격과정을 정리하면 공격자는 취약한 정상 웹 사이트의 권한을 탈취하여 페이지 변조를 통해 해당 취약점을 일으키는 페이지로 사용자를 이동시킨다. 해당 취약점을 이용하는 코드가 있는 웹페이지는 취약한 XML Core Services객체를 선언하여 웹 페이지의 DOM객체에 해당 객체를 연결시킨다. 만약 XML Core Services가 활성화 되어 있지 않다면 해당 취약점이 발생하지 않는다. 이후 Internet Explorer 프로세스의 Heap영역에 공격자가 의도한 악성 Shellcode를 할당한 후 img객체의 src속성에 공격자의 코드가 있는 힙 영역의 주소를 반복하여 써놓고 XML Services객체의 definition함수가 호출 될 때 사용할 객체의 로컬 변수 영역을 덮어씀으로써 공격자가 Heap에 위치시켰던 Shellcode가 사용자의 컴퓨터에서 실행이 된다. Shellcode는 추가적인 악성코드를 다운받아 실행하도록 하는 Downloader의 성격이 강하며, 사용자는 웹 페이지 방문만으로 악성코드에 감염되어 게임계정 및 개인정보가 탈취되거나, 공격자의 추가 공격을 할 수 있도록 백도어가 생성될 수도 있다.

현재 XML Core Services를 제공하는 msxmlx.dll은 Internet Explorer가 구동될 때 기본적으로 같이 로드되어 실행되는 라이브러리이기 때문에 Internet Explorer를 사용하는 모든 사용자가 이 위협에 노출되어 있다. 특히 방문자가 많은 사이트에 이러한 악성 웹 페이지가 있을 경우, 임시 패치를 하지 않은 모든 사용자가 감염될 위험이 있다.

아직 정식패치가 나오지 않는 시점에서 상당한 공격 성공률을 보일 것으로 예상되는바 신속한 Microsoft사의 정식패치가 필요한 상황이다.

3. 참 고 문 헌 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