An analysis of Microsoft Windows Vista's ASLR

Ali Rahbar <a.rahbar@sysdream.com>

www.sysdream.com

Note:

After working with Microsoft's security engineers on this topic this note has been added to reflect changes that have been introduced in Vista's RC1 and the reason why the entropy was so low in my tests.
Firstly I want to mention that in contrast to the 5472 build, stack randomization is not activated by default in RC1 and you should link your program with /dynamicbase to activate it.
Secondly, my analysis was based on the value of EBP at the creation of the process. On further inspection it turns out there is a second source of entropy. In the first stage of randomization which is analyzed in this article they only use 32 values for randomization. This is done to prevent excessive address space fragmentation. After that they randomize ESP. This randomization is processor dependent but on 32 bits processors they have 9 bits of entropy, which, with the abovementioned 4bits, gives us a total of 14 bits of entropy.

Ali Rahbar

10/5/2006

Since the release of the Beta 2 version of Windows Vista, Microsoft has added ASLR (address space layout randomization) to protect it from buffer overflows. ASLR is not new and has been available for a long time on other operating systems, but the advantage of Vista's ASLR is that it is activated by default.
The concept of ASLR is really simple; it changes the address space layout of a process to prevent predictable addresses that can be used in buffer overflow exploitations. For example, in a stack overflow exploitation the return address which is stored on the stack is overwritten by the address of a buffer which is controlled by the attacker. If the address of the buffer changes, the attacker will not be able to use it as a return address.
So the important point in an ASLR implementation is what is randomized and how it is randomized.
As pointed out by Michel Howard in his blog (http://blogs.msdn.com/michael_howard/archive/2006/05/26/608315.aspx), the heap and stack of all processes are randomized. A new PE header flag is used to determine whether EXEs and DLLs should be randomized. All EXEs and DLLs shipped with the OS are randomized. If a DLL which has its randomization flag set is loaded by an executable, the DLL will be randomized, whether the executable is randomized or not. This means that any DLL shipped with the OS (kernel32.dll, ntdll.dll…) will be randomized in all programs.
Now that we know what is randomized, we should figure out how they are randomized. In Windows Vista Beta 2 (32 bits) the randomization is done on 8 bits of the address, which gives us 256 possibilities. 256 is not a big number compared to other ASLRs like PaX. In some cases brute forcing the address is feasible.
I have tested the randomization of the stack on a sample program to determine if all 256 possibilities are used and whether they are used equally. I have used the following program for my tests:

```
#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);
}
```

After executing the program several times under Ollydbg, I have found that the third byte of all addresses are randomized. I have chosen to take the value of EBP at the entry point for analysis.
I have used PyDbg (by Pedram Amini) to create a little script that executes the program one million times and gets the value of EBP at each execution. The script is the following:

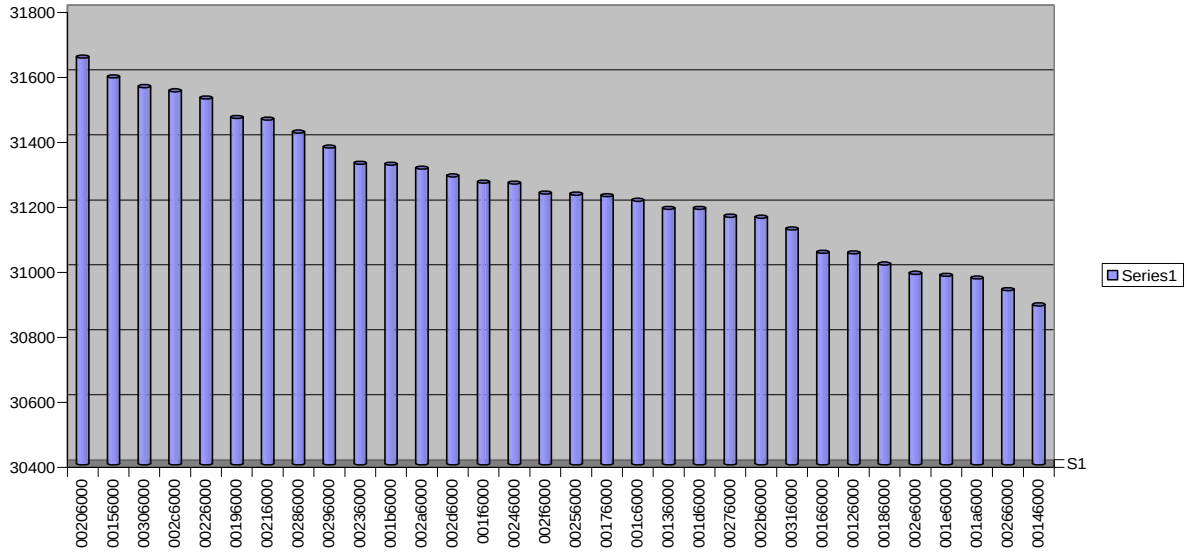
```
from pydbg import *
from pydbg.defines import *

def handler_breakpoint (db):
```

```
 # ignore the first windows driven breakpoint.
 if db.first_breakpoint:
  return DBG_CONTINUE

 a=db.stack_range()
 print '%x' % (a[1])
 db.terminate_process()
 del db

dbg=None
for i in range(10000):
 del dbg
 dbg=pydbg()
 dbg.set_callback(EXCEPTION_BREAKPOINT, handler_breakpoint)
 dbg.load("c:/buffer1.exe")
 dbg.bp_set(0x4012A9)
 pydbg.debug_event_loop(dbg)
```

Because of a memory leak I had to run this script 100 times to have one million EBP values. I didn't have the time to dig into it but I think the memory leak is in the Win32 python extension.
The big surprise came after counting the number of occurrences of each address. I have used a little program to count the number of occurrences of each address and to sort them. The result is quite impressive: Windows uses only 32 of 256 possibilities.
Here are the values of EBP (for my program):

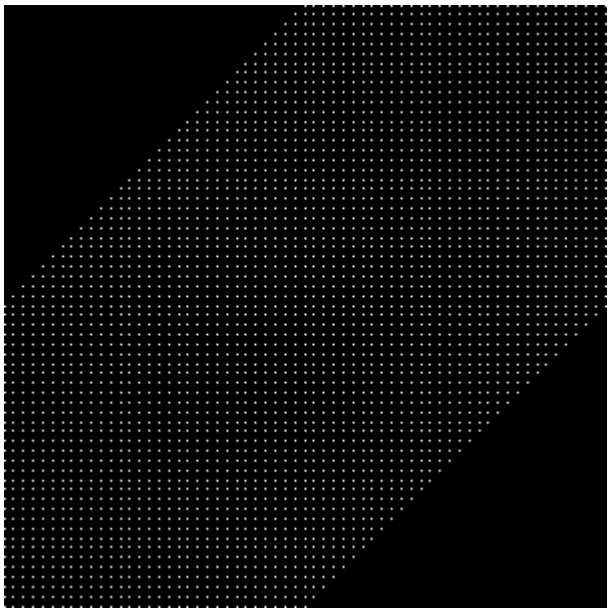
00206000
00156000
00306000
002c6000
00226000
00196000
00216000
00286000
00296000
00236000
001b6000
002a6000
002d6000
001f6000
00246000
002f6000
00256000
00176000
001c6000
00136000
001d6000
00276000
002b6000
00316000
00166000
00126000
00186000
002e6000
001e6000
001a6000
00266000

00146000
The following chart displays the number of times each value has been used. As you can see, the distribution between them has been done quite equally. The most frequent address has appeared 31653 times and the less frequent one 30890 times.

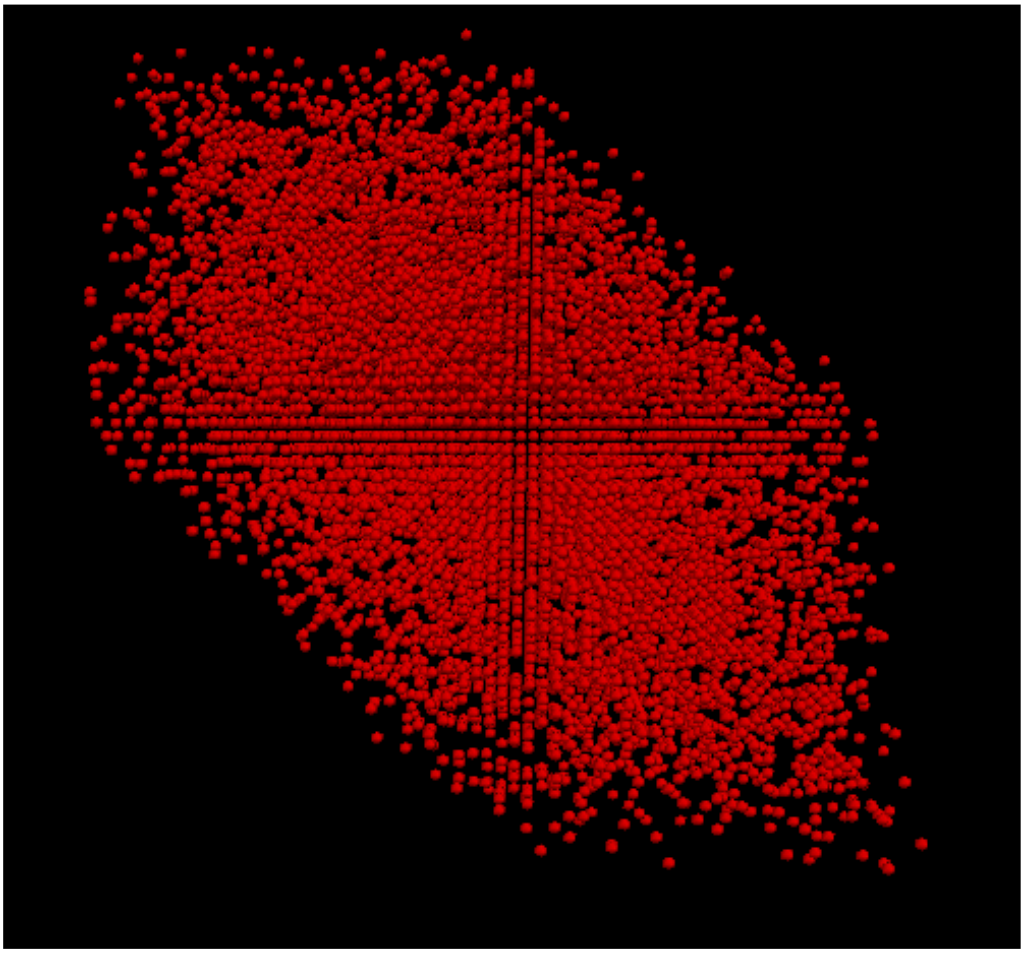


But 32 possibilities is not much, and for buffer overflow exploitation, in some situations it is really feasible to do a brute force on the 32 possible values. But why has Microsoft used only 32 out of 256 possibilities. The following 16x16 diagram shows the distribution of the values used by the ASLR in the space of 256 possibilities.

The white stripe represents used numbers and the black area represents other possibilities that have not been used by the ASLR.
To better see the dependencies between subsequent values, here is a 2D space phase diagram of these numbers:



In this diagram n stands for the input sets and x[i] = n[i-1]-n[i-2] ,y[i] = n[i]-n[i-1] which are the point's coordinates.
Here is the 3D representation in which x[i] = n[i-2]-n[i-3],y = n[i-1]-n[i-2] and z =n [i]-n[i-1]



As you have seen, the ASLR of Windows Vista Beta 2 is far from using all of its potential. I hope that this will be fixed in the final release.

At last, I would like to thank my friend Renaud Lifchitz who has helped me a lot on making these diagrams.

Ali Rahbar

09/11/2006

a.rahbar@sysdream.com