Address Space Layout Randomization

ASLR is a defense against all kinds of buffer overflows. This includes –

a. Code injection  
b. Return to libc attacks  
c. ROP attacks, etc.

It involves the randomization of the places the major components of the program (stack, heap, libraries) put in the memory.

For e.g. from a program’s perspective, there’s no requirement that the stack should always be at a particular location. The application program is concerned only the increment and decrement operations on the Stack Pointer (SP). So, the address of the stack can be changed without affecting the program.

The following components can be randomized –

a. Place stack at random location in the virtual memory space –

   The kernel can place the stack at a random location in the vm space, instead of the fixed location above the heap.

   Assuming 4kb pages, there are $2^{12}$ byte pages in a 4GB memory space. Most systems have a requirement that the stack has to be aligned at a page. So, there are $2^{32}/2^{12} = 2^{20}$ possible memory locations to load the stack into. In reality, due to other constraints, only $2^{16}$ locations are possible.

   Once this is done, the attacker can no longer guess the address of the stack – it’ll be difficult to guess the right location of the stack from the $2^{16}$ possible memory locations. So s/he can’t use the buffer overflow techniques to overwrite the stack. But return-to-libc attacks are still possible. However, passing arguments to the functions in libc (e.g. passing the cmd string “/bin/sh” to the system() function) become difficult.

   On possible way the attacker might try to circumvent this difficulty is by trying to fill up the heap with multiple copies of “/bin/sh”. This can be done by trying to log in hundreds of times with the user id “/bin/sh”. Then there is a good chance that a random pointer in the address space would point to the string “/bin/sh”.

b. The kernel can also randomize –

   i. Location of libraries/sections (again, has to be page-aligned)
   ii. Location of heap
iii. mmap locations

If the location of the libc library itself is not known, then attacks like return-to-libc can’t be carried out.

**Downsides**

The downsides of this is that it leads to the fragmentation of the VM address space.

In addition to this, there’s a major issue with the ASLR implementation on Windows. Unlike Linux where libraries are typically Position Independent (PIC code), on Windows some libraries must be loaded at a fixed address. Windows enables ASLR only for those applications which request for ASLR to be turned on. So the attacker can use ROP to search for gadgets in these non-randomized libraries.