How buffer overflow works?

- Stack smashing technique
- Assumption: attacker approximately knows the layout of the memory.

Idea behind ASR

- Uses the above mentioned assumption of buffer overflow attack
- Randomizes the addresses of various elements such as stack, heap, txt so that attacker is unable to specify the exact address in buffer overflow attack

How this is done

- Shift the start of the stack up or down.
- But Trampoline attack will still work against the randomized start of stack.
- So in order to avoid Trampoline, shift mmap and txt areas also
  - Shifting libs/mmap area is not hard because libraries are designed so that they are easy to shift
  - What about txt area? Hard to do but possible
  - But mmap, txt areas need to be aligned on page boundaries
Randomization possible in heap? Max 20 bits only

How the attacker exploited the fact that even though base address (start address) is randomized, the relative offsets are not randomized

- Find the address of sleep() function in libc using brute force
- Once found, calculate the offset by comparing the randomized sleep address with the standard sleep() function in libc
- Use this offset to compute the randomized address of system()

Implications

- 16 bits of randomness approximately equals 4 minutes
- Attacker may not need to guess all randomness simultaneously

Fixes

- 64-bit architecture with 64-bit address space
- Run-time randomization

Internal randomization (compile-time randomization)

- stack
  - Local variable ordering
  - Intraframe & interframe padding
  - Frame ordering
  - Parameter ordering – very difficult but possible
- Code segment
  - Function ordering
  - Padding
  - Instruction choices
  - Basic blocks
  - Polymorphic code?
  - Instruction set randomization
- Heap
  - Inter-allocation padding
  - ordering