

# Spring 2009 - CSE509 - System Security

## Lecture Notes – 03/20/2009

### Address Space Randomization

Address space randomization (ASR) is one of the techniques used to prevent buffer over flow attacks.

#### **Assumption**

For an attacker to successfully perform the buffer overflow attacks, he should know the process's memory address space layout. This is an easy task, as the attacker can use the same hardware and software as is used in the system he is trying to compromise.

#### **How could this be prevented?**

Take away this knowledge (process address space layout) from the attacker. This can be accomplished by randomizing the process memory layout. Few techniques can be used for this,

- Shift the start of the stack.
- Shift the text and mmap-ed (libraries) areas. This is to avoid attacks through trampolines.

Even though the start of these sections are randomized, on a 32-bit system the attacker uses a brute-force attack trying out all possible addresses and in a couple of minutes he will be able to compromise the system. In reality, the attacker doesn't have to try all possible  $2^{32}$  addresses, instead he just has to try out only  $2^{16}$  addresses for the reasons given below.

- Since the mmap-ed area has to be aligned to page boundaries (assuming page size as 4K), the attacker can ignore the last 12 bits and try out with setting that to zeros.
- The attacker can also ignore the first 4 bits, as a small change to these most significant bits would move (either up or down) the mmap-ed section with greater magnitude.



Thus the attacker is left with 16 bits ( $32 - 12 - 4$ ) and that is the effective randomization that he has to try. The paper\* claims that with 16 bits of randomization space, on an average the attacker can exploit the system in about 4 minutes.

Looking at this substantiates that 32-bit systems are more vulnerable to these kinds of attacks and thus moving on to 64-bit systems can be a cure. It actually is, as it exponentially increases the randomization space mitigating the attacker's capabilities.

The techniques that we have discussed so far are external randomizations ie., moving the sections stack, text or heap as a whole. Another possibility is to do internal randomization.

## Internal Randomizations

Internal randomizations are can performed inside the stack, heap or the text (code) section.

- Stack
  - Local variable ordering
  - Inter-frame padding
  - Intra-frame padding
  - Frame ordering
  - Parameter ordering
- Code Section
  - Function ordering
  - Padding
  - Basic blocks ordering
  - Instruction choices
- Heap
  - Inter-allocation padding
  - Ordering

Mostly these randomizations are done at the compile time by the compiler. From the implementation point of view, implementing internal randomizations are a little demanding compared to the previous techniques.

There is also techniques called re-randomization, ie, to perform randomization at every invocation of the application or re-randomizing at the run-time. But the paper\* says even these techniques just add a single bit to the total randomization space.

*Why is address space randomization so successful with OS vendors?*

This is because, ASR is the only technique in the hands of OS vendors to counter attack these kind of exploits. So the OS vendors are trying hard to come up with their best possible ASR technique and to an extent has been successful so far.

[\*] - H. Shacham, M. Page, B. Pfaff, E. Goh, N. Modadugu, D. Boneh, On Effectiveness of Address-Space Randomization.

[Lecture notes collated by Arun Ponniah Sethuramalingam]