Recall: Steps of Stack Smashing Attack

1. Find a buffer overflow vulnerability in the program
2. Inject shellcode into a known memory address
3. Exploit the buffer overflow vulnerability to overwrite EIP with the shellcode address.
4. Return from the vulnerable function.
5. Start to execute the shellcode.

Key insight of defense:

- Make some critical steps more difficult or even impossible to achieve.
- The attacker can only crash the system, but not hijack the control flow to execute arbitrary code.
- This is possibly denial-of-service attacks. Availability is not the main consideration of our threat model. Integrity is more important.

Recall: Steps of Stack Smashing Attack

1. Find a buffer overflow vulnerability in the program
2. Inject shellcode into a known memory address
3. Exploit the buffer overflow vulnerability to overwrite EIP with the shellcode address.
4. Return from the vulnerable function.
5. Start to execute the shellcode.

Solution:

- Address Space Layout Randomization (ASLR)

Address Space Layout Randomization (ASLR)

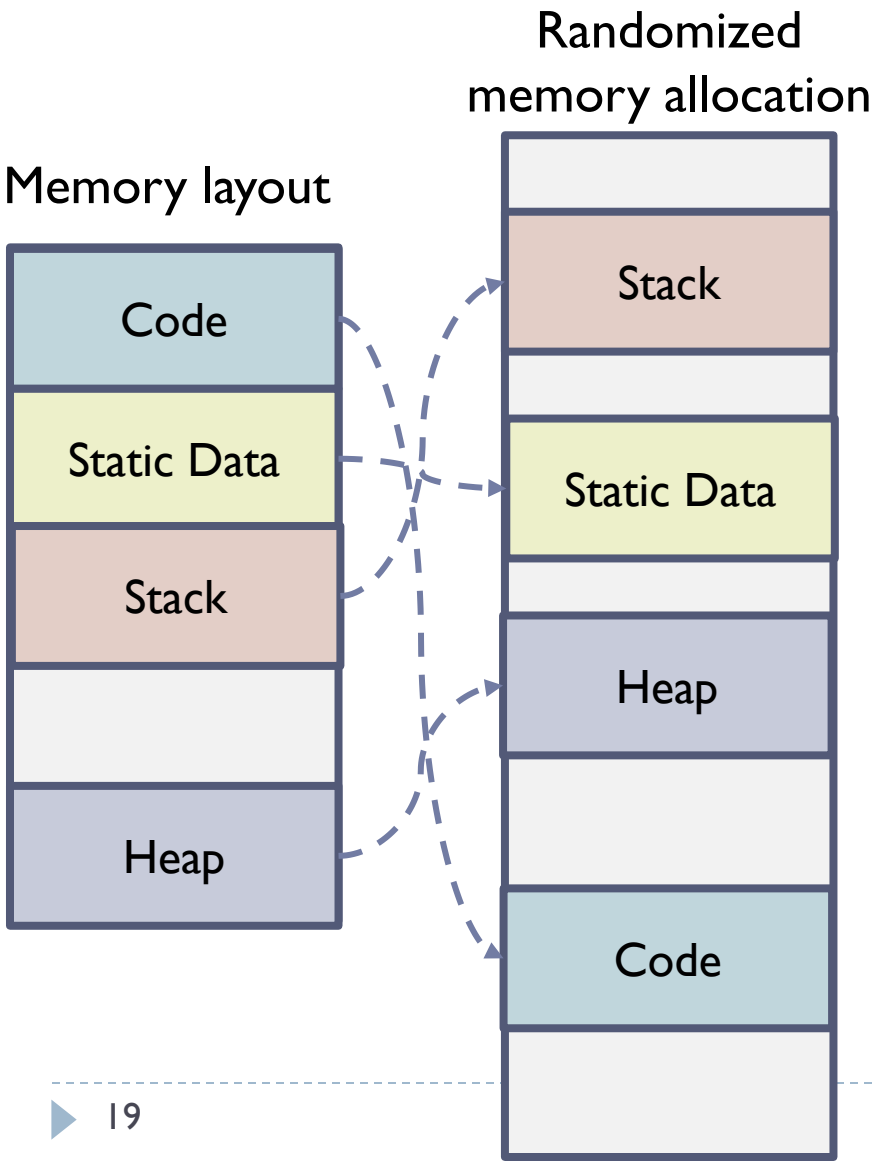
Key idea

- Put the segment of each memory region (data, code, stack, heap, etc.) in a random location every time the program is launched.
- Make it harder for the attacker to get the address of his malicious function
- Within each segment, relative addresses are the same
- Program remain correct if base pointers of these regions are set up correctly
- No performance overhead

ASLR is practical and widely deployed in mainstream systems

- Linux kernel since 2.6.12 (2005+)
- Android 4.0+
- iOS 4.3+ ; OS X 10.5+
- Microsoft since Windows Vista (2007)

ASLR Example




```c
#include <stdio.h>
#include <stdlib.h>
void main() {
    char x[12];
    char *y = malloc(sizeof(char)*12);
    printf("Address of buffer x (on stack):
0x%x\n", x);
    printf("Address of buffer y (on heap) :
0x%x\n", y);
}
```

```
$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
$ a.out
Address of buffer x (on stack): 0xbffff370
Address of buffer y (on heap) : 0x804b008
$ a.out
Address of buffer x (on stack): 0xbffff370
Address of buffer y (on heap) : 0x804b008
```

```
$ sudo sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
$ a.out
Address of buffer x (on stack): 0xbf9c76f0
Address of buffer y (on heap) : 0x87e6008
$ a.out
Address of buffer x (on stack): 0xbfe69700
Address of buffer y (on heap) : 0xa020008
```

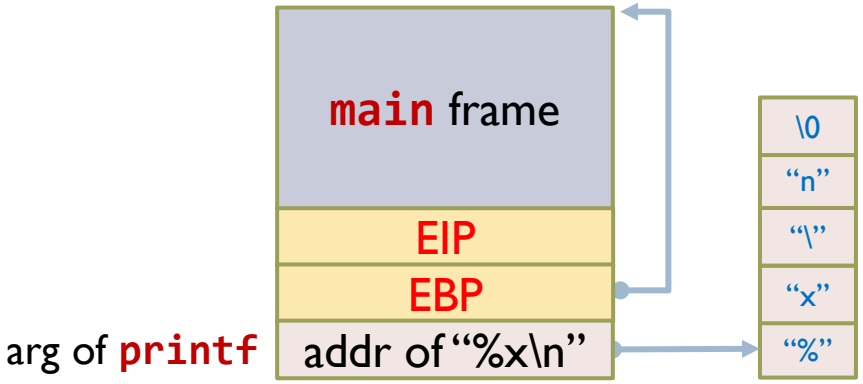
Insecurity of ASLR

Attacker gets the base address of stack first. As the relative addresses within the stack are normally fixed, the attacker can compute the addresses of any data in the stack.

- The attacker can use brute-force technique to guess the base address.
- Format string vulnerability allows the attacker to print out base pointer from the stack (%x).

```
int main(void){

    printf("%x\n");
    return 0;
}
```



Recall: Steps of Stack Smashing Attack

1. Find a buffer overflow vulnerability in the program
2. Inject shellcode into a known memory address
3. Exploit the buffer overflow vulnerability to overwrite EIP with the shellcode address.
4. Return from the vulnerable function.
5. Start to execute the shellcode.

Solutions:

- StackGuard
- Shadow Stack
- StackShield
- PointGuard
- Pointer Authentication

StackGuard

Key insight

- It is difficult for attackers to only modify the return address without overwriting the stack memory in front of the return address.

Steps

- Embed a canary word next to the return address (EIP) on the stack whenever a function is called.
  - Canary value needs to be random and cannot be guessed by attacker.
- When a stack-buffer overflows into the function return address, the canary has to be overwritten as well
- Every time the function returns, check whether canary value is changed.
- If so, someone is possibly attacking the program with stack-buffer overflows, and the program will be aborted.

First introduced as a set of GCC patches in 1998