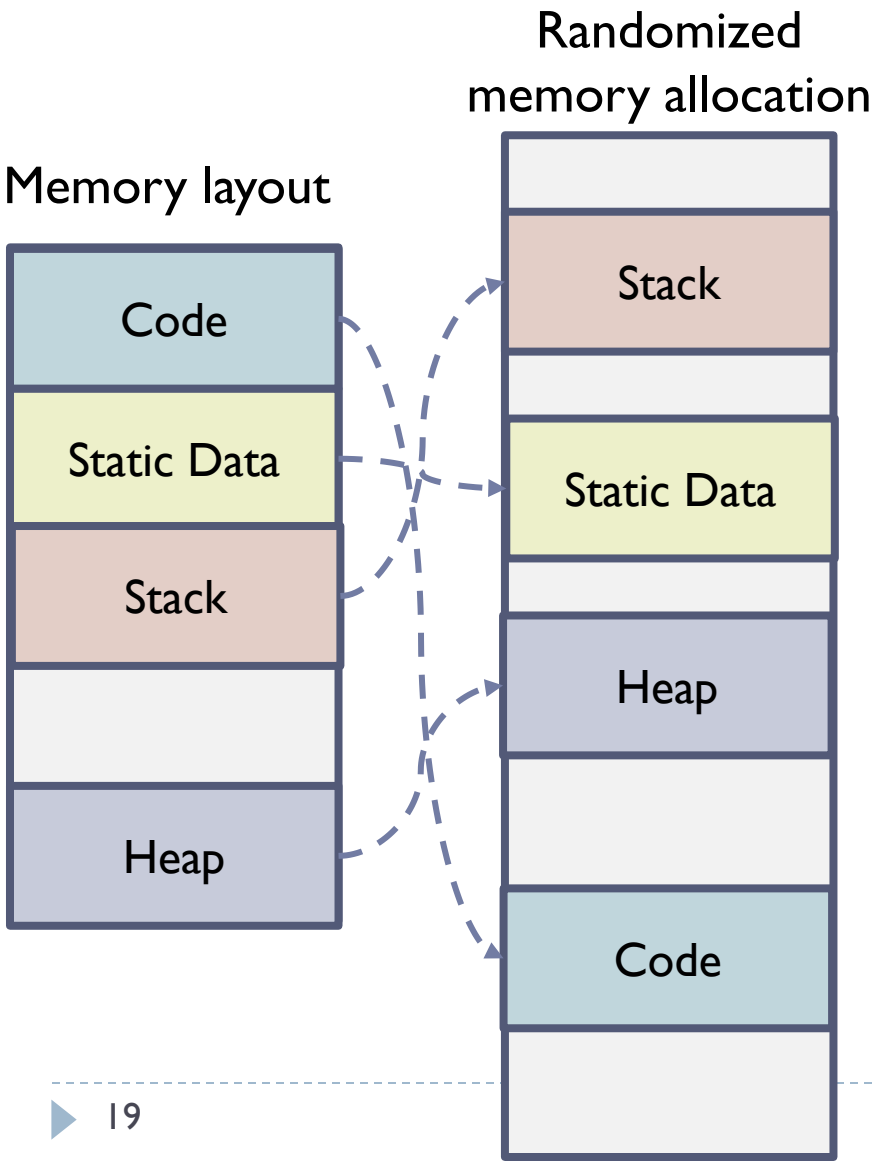
ASLR Example




```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
```

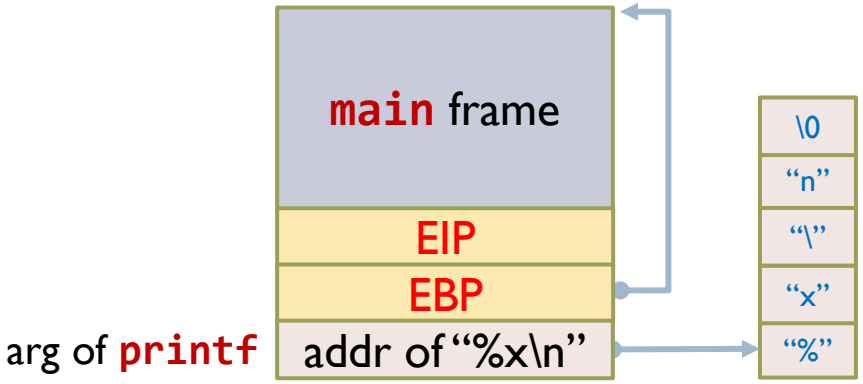
```
$ sudo sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
$ a.out
Address of buffer x (on stack): 0xbf9c76f0
Address of buffer y (on heap) : 0x87e6008
$ a.out
Address of buffer x (on stack): 0xbfe69700
Address of buffer y (on heap) : 0xa020008
```

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

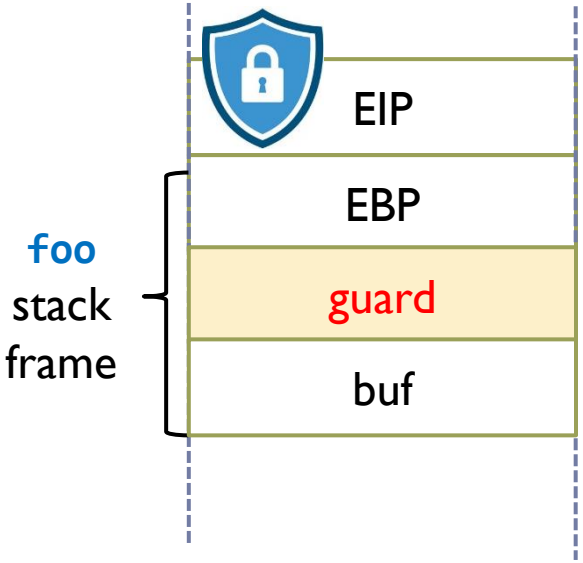
- Embed a canary word next to the return address (EIP) on the stack whenever a function is called.
  - Canary value needs to be random and cannot be guessed by attacker.
- When a stack-buffer overflows into the function return address, the canary has to be overwritten as well
- Every time the function 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

How does StackGuard Work

```
void foo(char *s) {


  char buf[16];
  strcpy(buf,s);



}
```

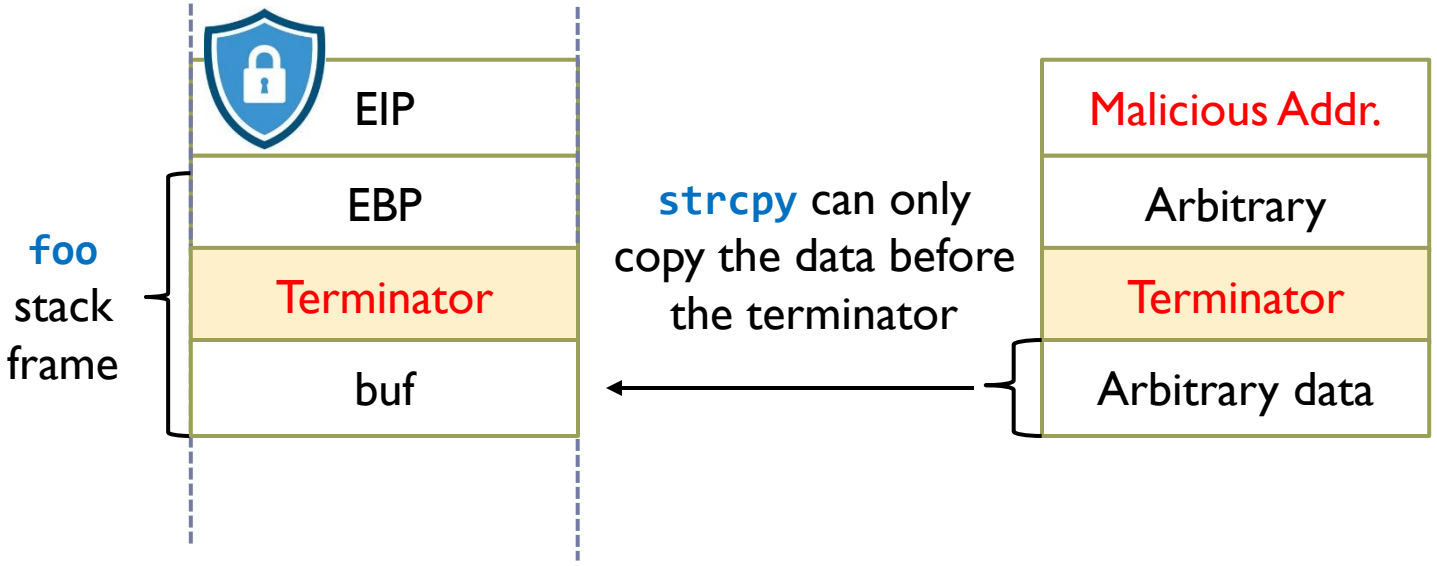
```
int *secret = malloc(size_of(int));
*secret = generateRandomNumber();

void foo(char *s) {
  int guard;
  guard = *secret;

  char buf[16];
  strcpy(buf,s);

  if (guard == *secret)
    return;
  else
    exit(1);
}
```



An Alternative Canary Type

Terminator canary

- Canary = {\0, newline, linefeed, EOF}
- String functions will not copy beyond terminator
- Attacker cannot use string functions to corrupt stack.



Insecurity of StackGuard

Attacker can obtain the canary's value, which will be used to overwrite the canary in the stack without changing the value.

- Format string vulnerability allows the attacker to print out values in the stack (%x).
- The attacker can use brute-force technique to guess the canary.

Attacker can overwrite the return address in the stack without touching the canary.

- Format string vulnerability allows the attacker to write to any location in memory, not need to be consecutive with the buffer (%n).
- Heap overflows do not overwrite a stack canary.

Shadow Stack

Keep a copy of the stack in memory

- On function call: push the return address (EIP) to the shadow stack.
- On function return: check that top of the shadow stack is equal to the return address (EIP) on the stack.
- If there is difference, then attack happens and the program will be terminated.

Shadow stack requires the support of hardware

- Intel CET (Control-flow Enforcement Technology):
  - New register SSP: Shadow Stack Pointer
  - Shadow stack pages marked by a new "shadow stack" attribute:
  - only "call" and "ret" can read/write these pages