BUFFER OVERFLOW
BUFFЕR OVERFLOW

- A common security vulnerability

- Root cause
  - Unsafe programming languages
  - The problem would disappear if we would write correct code

- What areas of process memory are vulnerable to a buffer overflow?
  - Stack
  - Heap
  - Code/Data
STACK SMASHING ATTACK

- A specific kind of buffer overflow attack

- How does it work?
  - During a function call, the return address is pushed on the stack
  - An attacker overflows a buffer (local variable)
  - The return address on the stack is overwritten to point to an existing function or to injected code
  - During the function return the instruction pointer is set to the new return address value stored on the stack, not the original return address that was pushed on the stack as the function was called
VULNERABLE CODE EXAMPLES

This code snippet caused the Morris Worm (1988)

```c
char buf[20];
gets(buf);
```
VULNERABLE CODE EXAMPLES

```c
void foo(char *input) {
    //make a local working copy
    char buf[1024];
    strcpy(buf, input);
}
```
LIMITATIONS

- Usually there is only one write operation that is vulnerable
  - The attacker has one operation to overwrite the return address
  - The stack frame is usually corrupted so that the program crashes sometime after the buffer is overflowed
    - But the attack may be executed before the crash occurs
  - Remote attacker doesn’t know the exact address location of the injected attack code
    - NOP Sled helps create a window of opportunity
**QUESTIONS ON STACK SMASHING**

- How does the stack normally operate during a function call/return?
  - Where is the stack in memory?
  - How do the base pointer (ebp) and stack pointer (esp) work?
  - How are local variables placed on the stack?

- Describe how an attacker can inject code on the stack

- What is a NOP sled and how/why is it used in a stack smashing attack?

- What are the requirements for the format of the injected code?
DEFENSES

- Write correct code
  - Avoid vulnerable functions
  - Audit code – use analysis tools
  - Fuzz testing

- Non-executable buffers
  - Kernel patches make the stack non-executable

- Array bounds checking
  - Compile time or run-time checks
  - Use a type-safe language

- Code pointer integrity checking
  - Detect when a pointer is corrupted
  - Canaries and pointer checking

- Address space randomization (ASLR)
How does a canary prevent a stack smashing attack?

- Canary types
  - Terminator canary
  - Random canary
  - XOR canary
QUESTIONS

- What are the approaches to defend against a buffer overflow attack?
  - What are the pros/cons of each?
- Is a buffer overflow only useful for a remote attack?
- (True or False) Making the stack non-executable makes a stack smashing attack impossible?
- (True or False) If your web server is written in Java, it is not vulnerable to a stack smashing attack?
- What is the principle of least privilege and how does it relate to buffer overflow attacks?
Three main integer manipulations that can lead to security vulnerabilities:
- Overflow and underflow
- Signed vs. unsigned errors
- Truncation

Reviewing Code for Integer Manipulation Vulnerabilities