1988  **Morris Worm**  ... estimated 10% penetration

2001  **Code Red**  ... 300,000 computers breached

2003  **Slammer/Sapphire**  ... 75,000 infections in 10 min.

2005  **Zotob**  ... exploits MS Windows plug and play

2008  **Conficker**  ... European military computers quarantined, flights grounded, naval operations disrupted

**Public Enemy #1:**

This is especially a problem in C & C++

**A Simple C/C++ Example:**

What happens when the assignment executes?

...but how can this be exploited?
C++ Example 1:

```cpp
#include<stdio.h>
using namespace std;

static char input[5];
static char stuff[10];

int main() {
    gets(input); //gets reads a line of text from the command line
    puts(input); //puts outputs a string on a separate line
    puts(stuff);
}
```

C++ Example 2:

```cpp
bool IsPasswordOkay() {
    char password[12];
    gets(password);
    if (!strcmp(password,"goodpass"))
        return true;
    else
        return false;
}

int main() {
    bool status;
    puts("Enter Password:");
    status=IsPasswordOkay();
    if (status==false) {
        puts("Access denied");
        return -1; //failure exit
    } else
        puts("Access granted");
    return 0; //normal exit
}
```
What things are stored in a runtime stack?

For example, assume foo1 was the first function called. foo1 called foo2, which in turn called foo3.

For example, assume foo1 was the first function called. foo1 called foo2, which in turn called foo3.

```
bool IsPasswordOkay() {
    char password[12];
    gets(password);
    if (!strcmp(password, "goodpass"))
        return true;
    else
        return false;
}

int main() {
    bool status;
    puts("Enter Password:");
    status=IsPasswordOkay();
    if (status==false) {
        puts("Access denied");
        return -1;  //failure exit
    } else
        puts("Access granted");
    return 0;  //normal exit
}
```

Assume this input:

```
123456789ABCDEFGHIJK
```

<table>
<thead>
<tr>
<th>password (12 bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frame pointer to main (4 bytes)</td>
</tr>
<tr>
<td>Return address in main (4 Bytes)</td>
</tr>
<tr>
<td>status (4 bytes)</td>
</tr>
<tr>
<td>parameter storage for main()</td>
</tr>
<tr>
<td>Frame pointer to OS</td>
</tr>
<tr>
<td>Return address in OS</td>
</tr>
</tbody>
</table>

Runtime Stack
How is memory used by an executing program?

- static memory
- runtime stack
- heap
- code

Buffer overflow often causes breaches via ...

- __________ - the overflow corrupts the runtime stack
- __________ - the overflow corrupts a pointer

```c
void foo(void *arg, int len) {
    int value = something;
    int *ptr = somethingElse;
    memcpy(buff, arg, len); //potential overflow
    *ptr = value; //attacker can control both data 
    ... // and where it is written
    return;
}
```

```c
void goodFunction() { ... }
void foo(const char argv[]) {
    static char buff[10];
    static void (*funcPtr) () = &goodFunction;
    strcpy(buff, attackStr, strlen(argv)); //potential overflow
    (void) (*funcPtr)(); //attacker can control invoked function
    ... return;
}
```
Looking a little closer...

The best way to stop stack smashing and pointer subterfuge is to stop the cause ➔ buffer overflow.

Buffer overflows should not go undetected.

Using null string termination is a bad idea.

Mitigations

- Read item by item and check for length.

- Avoid C functions that are suspect
  
  - `strcpy()`, `strcat()`, `gets()`, `scanf()`
  - `sscanf()`, `sprintf()`, `fscanf()`
  - `vfscanf()`, `vprintf()`, `vscanf()`
  - `vsscanf()`, `streadd()`, `strecpy()`
  - `strtrns()`

- Use safer C functions
  
  - `strncpy()` instead of `strcpy()`
  - `fgets()` instead of `gets()`

- Use canaries
More Mitigations

• Use safe C libraries, such as OpenBSD `strlcpy()` or C++ `std::string`

• Perform code reviews (look for unsafe functions)

• Security testing

• Use static or dynamic analysis tools

• Use safer programming language

What do you know about “your” programming language(s)?

Java Example 1:
```java
int k = Integer.MAX_VALUE;
System.out.println( k );
k++;
System.out.println( k );
```

Java Example 2:
```java
int n = 0x102;
System.out.println( n );
System.out.println( (byte) n );
```
What do you know about “your” programming language(s)?

Java Example 3:
```
int j = 0x80;
System.out.println( j );
j = j >>> 2;  //shift bit string right by 2 bits (unsigned)
System.out.println( j );

byte b = (byte)0x80;
b = (byte)(b >>> 2);  //shift bit string right by 2 bits (unsigned)
System.out.println( b );
```

Integer Vul Mitigations

- Use larger data type than needed
  - byte ➔ short
  - short ➔ int
  - int ➔ long

- Check ranges
  ```java
  int nextInt = myInt + 1;
  if (nextInt > myInt)
    myInt = nextInt;
  else
    // handle error exception
  ```

- Use trusted integer values and safe operations
- Use unsigned types for indices, size and loop counters