University of BRISTOL

COMS20012: Memory Corruption

Joseph Hallett

bristol.ac.uk

So what is all this about?

- Software has bugs
  - Sometimes we can use these bugs to violate security principles
- Memory corruption bugs (and memory safety)
  - Corrupt the memory of a program to violate security principles
  - Can lead to:
    - arbitrary read
    - arbitrary write
    - control flow hijack
    - control flow corruption

Next few lectures

- We're going to go over some memory corruption bugs and how to exploit them to cause weird behaviour
  - …but first a bit more on memory corruption *in general*

Pointers

- Pointers allow you to refer to (semi) arbitrary memory addresses in most programming languages
  - Well… in C
- Some languages claim not to have them (e.g. Java)
  - Not strictly true… just not usually as easy to abuse as Cs
  - (no arithmetic, no arbitrary addresses)
- To introduce a bug…
  - Get a pointer pointing somewhere it shouldn't

Example

```
#include <stdio.h>
int main(void) {
  int x;
  int buffer[4];

  x = 0;
  printf("x = %d\n", x);
  buffer[-2] = 1;
  printf("&x       = %p\n", &x);
  printf("buffer   = %p\n", buffer);
  printf("buffer-2 = %p\n", buffer-2);
  printf("x = %d\n", x);
  return 0;
}
```

Obviously a warning…

```
cc  -I/opt/homebrew/opt/openjdk/include   example.c   -o example
example.c:8:3: warning: array index -2 is before the beginning of the array [-Warray-bounds]
  buffer[-2] = 1;
  ^      ~~
example.c:4:3: note: array 'buffer' declared here
  int buffer[4];
  ^
1 warning generated.
```

(though this is technically allowed by the C standard so not an error 🤦)

But without assigning to `x`…

```
joseph@Dingus-Mingus Desktop % ./example
x = 0
&x        = 0x16bcff6e0
buffer    = 0x16bcff6e8
buffer-2 = 0x16bcff6e0
x = 1
```

There is a whole lot more fun to be had

- What happens if you go beyond a local array's end?
  - You can get arbitrary execution…
  - (See *Smashing the stack for fun and profit)*
  - Sometimes called a *spatial error*
- What happens if you use memory after you've freed it?
  - You can get an arbitrary write…
  - (See the *malloc maleficarum)*
  - Sometimes called a *temporal error*
- What happens if you can pass an arbitrary string to *printf*?
  - You can get an arbitrary write…
  - (See *format string vulnerabilities)*
  - WTF C?

The problem with C (and C++)

- C was designed to write operating systems
- Programmers were expected to know what they were doing
  - i.e. if you're going off the end of an array its deliberate and not a mistake

- If you don't know what you're doing C can be dangerous…
  - There is no type safety like Java or Haskell
  - You can do strange maths with pointers
  - Semantics are weird and surprising

  - Programmers have been trained to ignore warnings…

How do we fix this?

- Short term:
  - Don't teach programmers unsafe practice
  - Listen to your compiler (`–Wall –Wextra –Weverything –Werror`)
- Longer term
  - Maybe we should make it harder to do dangerous things?
  - Lots of legacy C code out there (some even pre-ANSI C)
  - Do you really need pointers to write fast applications?
  - Will the cost of rewriting 30 year old code in a memory safe language outweigh the bugs you'll inevitable introduce rewriting it?