**INTRODUCTION**

**Common Tools**

* Code Review
  + Source Insight
  + VIM + CTAGS + CScope
  + OpenGrok
* Reverse Engineering
  + IDA Pro (free version)
* Fuzzing
  + Python or Ruby or a framework written in the two languages
* Debugging
  + GDB

**Threat Modelling / Attack Surface**

* A set of possible attacks to consider:
  + Where is it exposed to untrusted data?
  + What could a potentially bad person manipulate?
  + What has the developer assumed? (Assumptions are bad)

**Risk Assessment** (Common way govt/bsns assess the actual “threat” of a vulnerability in a system)

* Two factors are used

1. Likelihood / Difficulty: How likely the event is to happen
2. Impact: What’s the damage if it happens?

* Good risk assessments, in terms of vulnerabilities are based on:
  + Technical difficulty: Days required to develop an exploit
  + Resources required: Extremely skilled people? Access to local network? 100’s of good machines to break the crypto? Profit of attacker?



**MEMORY**

CPU’s at a low level simply see a long strip of main memory

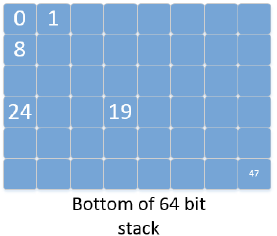
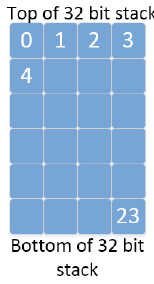
CPU’s have a limited number of registers, not considered main memory

Memory is more like multiple 3D objects (stick of RAM of various sizes) rather than 1D / 2D objects. This doesn’t really matter as the processor just accesses it all by address.

32 bit stacks = 4 bytes wide

64 bit stacks = 8 bytes wide

Visualise this in 2 dimensions, where width = #bytes of the basic unit we use



|  |  |
| --- | --- |
| Char (signed + unsigned) = 1 byte | Short (signed + unsigned) >= 2 bytes |
| Int (signed + unsigned) = 4 bytes | Long (signed + unsigned) >= 4 bytes |

**Unsigned ints**

* Unsigned int values are encoded in pure binary form
* Converting from binary 🡪 decimal:
  + Value of each bit position n, multiplied by **2n**
  + Examples:  
    0001 1011 🡪 2^4 + 2^3 + 2^1 + 2^0 = 27  
    0000 1111 🡪 2^3 + 2^2 + 2^1 2^0 = 8 + 4 + 2 + 1 = 15

**Two’s Complement**

* Signed values on 32bit are stored using two’s complement
* Example: Reading 1 vs -1. Positive 1 = 0000 0001. Therefore -1 =
  + Step 1: Invert all bits = 1111 1110 (hex: FFFF FFFE)
  + Step 2: Add one = 1111 1111 (hex: FFFF FFFF)

**Pointers**

* ­­­Pointers are variables which contain the address of where they point to.

**Endianess**

* WTF read more about this on the internet… the slide makes no sense

**Arrays**

* Arrays are just lined up lists of the original type (e.g. int array = list of ints)
* Strings are just character arrays