~“I have a bad feeling about this.”~

— Star Wars, Episode k
Data, Message & Memory Integrity

- **Anti-Patterns for Data Integrity:**
  - No checks on memory data
    - Program image and configuration
    - RAM and other data integrity
  - No end-to-end message checks
    - Using checksum instead of CRC

- **Memory & data integrity**
  - Detecting data corruption:
    - Mirroring, Parity & SECMED codes, Checksum, CRC
    - If data word consistent with error code, then no *detectable* error
    - Random hash as a starting point: random k-bit error code by chance misses $1/2^k$ errors
  - Malicious faults require cryptographically strong integrity check
    - All error codes discussed here are easy to attack
Sources of Data Faults

- **Hardware faults**
  - Network message bit flips
  - Bad EEPROM/Flash writes
  - “Bit rot” (storage degrades over time)

- **Single event upsets: Soft Errors**
  - Affect both memory & CPU logic
  - Error detecting codes usually don’t help with CPU logic faults!

- **Software corruption**
  - Bad pointers, buffer overflow, etc.
Smaller Dataword Integrity Mechanisms

- **Key term: Hamming Distance (HD)**
  - Smallest # of bit flips possibly undetected
  - Flips across data value and error code
  - Higher HD is better (more errors detected)

- **Parity: detects single bit errors (HD=2)**
  - Store one bit that holds XOR of all bits

- **Mirroring (HD=2, but cheap computation)**
  - Store data twice: plain and inverted bits
    - E.g.: 0x55 $\rightarrow$ {0x55, 0xAA} two-byte pair

- **SECDED: Single Error Correction, Double Error Detection**
  - Use a Hamming Code + parity to give HD=3
  - Cost approximately $\log_2$ number of data bits

<table>
<thead>
<tr>
<th>HD</th>
<th>Flips Detected</th>
<th>Flips Undetected</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>None</td>
<td>1+</td>
<td>No Error Code</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2+</td>
<td>Parity, Checksum, Mirroring, Long CRC</td>
</tr>
<tr>
<td>3</td>
<td>1-2</td>
<td>3+</td>
<td>SECDED, Good CRC, Short Fletcher</td>
</tr>
<tr>
<td>4</td>
<td>1-3</td>
<td>4+</td>
<td>Good CRC</td>
</tr>
<tr>
<td>5+</td>
<td>HD-1</td>
<td>HD+</td>
<td>Good CRC</td>
</tr>
</tbody>
</table>
Checksum for Larger Data Structures

■ “Add” up all the data bits
  ● XOR all data words (HD=2)
    - Detects 1-bit errors
  ● 2’s complement addition (HD=2)
    - Detects 1-bit and most 2-bit errors
  ● 1’s complement addition (HD=2)
    - Wraps carry bit, so slightly better

■ Complex checksums:
  ● Fletcher checksum (HD=2, HD=3)
    - Keeps two running 1’s comp. sums
    - HD=3 at short lengths, HD=2 at long lengths
  ● Adler checksum (HD=2, HD=3)
    - Uses prime moduli counters
    - Usually Fletcher is better (and faster)

Cyclic Redundancy Check (CRC)

The mechanism:
- Shift and XOR of selected feedback bits
- Accumulated residue in shift register is the CRC "checksum" value

The math:
- The data and the feedback bit pattern are both binary coefficient polynomials
- Error code is remainder from polynomial division of data by feedback over GF(2)

Feedback polynomial selection matters
- Some popular polynomials are poor choices, including international standards(!)
- Some rules of thumb are misguided (e.g., (x+1) divisibility for high HD)
- Best polynomials are found via brute force search of exact evaluations

Example Feedback Polynomial:
0xB41 = x^{12} + x^{10} + x^9 + x^7 + x + 1 ("+1" is implicit in hex value)
= (x+1)(x^3 + x^2 + 1)(x^8 + x^4 + x^3 + x^2 + 1)
Factor of (x+1) $\Rightarrow$ implicit parity (detects all odd errors)
Best Practices For Data Integrity

Ensure sufficient data integrity
- CRC on network packets
- Periodic CRC on flash/EEPROM data
- Appropriate memory integrity check on RAM

Pitfalls:
- Assuming mirroring is enough
  - What about data on stack?
  - What about data inside operating system?
- Assuming memory data integrity is all you need
  - What about corrupted calculations?
- Using a checksum when you should use a CRC
- Many subtle pitfalls for the unwary. See FAA report: https://goo.gl/uKFmHr
nano? REAL PROGRAMMERS USE emacs

HEY. REAL PROGRAMMERS USE vim.

WELL, REAL PROGRAMMERS USE ed.

NO, REAL PROGRAMMERS USE cat.

REAL PROGRAMMERS USE A MAGNETIZED NEEDLE AND A STEADY HAND.

EXCUSE ME, BUT REAL PROGRAMMERS USE BUTTERFLIES.

They open their hands and let the delicate wings flap once.

The disturbance ripples outward, changing the flow of the eddy currents in the upper atmosphere.

Which act as lenses that deflect incoming cosmic rays, focusing them to strike the drive platter and flip the desired bit.

Nice. ‘Course, there’s an Emacs command to do that.

Oh yeah! Good ol’ C-x M-c M-butterfly...

DAMMIT, EMACS.
Digital Data

The great thing about digital data is that it never degrades.

Hard drives fail, of course, but their bits can be copied forever without loss.

Film degrades, paint cracks, but a copy of a century-old data file is identical to the original.

If humanity has a permanent record, we are the first generation in it.

Title text: "If you can read this, congratulations the archive you’re using still knows about the mouseover text!"