Outline for May 5, 2005

Reading: §15, §18

Discussion

Ever wonder why people need to write program specifications and requirements carefully? The answer is that those specifications seem to live forever. Here's an example:

The United States Standard railroad gauge (distance between the rails) is 4 feet, 8.5 inches. That's an exceedingly odd number. Why was that gauge used? Because that's the way they built them in England, and the United States railroads were built by English expatriates. Why did the English people build them like that? Because the first rail lines were built by the same people who built the pre-railroad tramways, and that's the gauge they used.

So, why did "they" use that gauge? Because the people who built the tramways used the same jigs and tools that they used for building wagons, which used that wheel spacing. Okay! Why did the wagons use that odd wheel spacing? Well, if they tried to use any other spacing the wagons would break on some of the old, long distance roads, because that's the spacing of the old wheel ruts.

So who built these old rutted roads? The first long distance roads in Europe were built by Imperial Rome for the benefit of their legions. The roads have been used ever since. And the ruts? The initial ruts, which everyone else had to match for fear of destroying their wagons, were first made by Roman war chariots. Since the chariots were made for or by Imperial Rome, they were all alike in the matter of wheel spacing.

Thus, we have the answer to the original questions. The United States standard railroad gauge of 4 feet, 8.5 inches derives from the original specification for an Imperial Roman army war chariot. Specifications and bureaucracies live forever. So, the next time you are handed a specification and wonder what horse's rear came up with it, you may be exactly right, because the Imperial Roman chariots were made to be just wide enough to accommodate the back-ends of two war horses.

— source unknown, but circulated on USENET and other message systems

Outline

1. Capabilities
   a. Capability-based addressing: show picture of accessing object
   b. Show process limiting access by not inheriting all parent's capabilities
   c. Revocation: use of a global descriptor table
2. Privilege in Languages
   a. Nesting program units
   b. Temporary upgrading of privileges
3. Lock and Key
   a. Associate with each object a lock; associate with each process that has access to object a key (it's a cross between ACLs and C-Lists)
   b. Example: use crypto (Gifford). X object enciphered with key K. Associate an opener R with X. Then:
   OR-Access: K can be recovered with any D_i in a list of n deciphering transformations, so
   $R = (E_1(K), E_2(K), ..., E_n(K))$ and any process with access to any of the D_i's can access the file
   AND-Access: need all n deciphering functions to get K: $R = E_1(E_2(...E_n(K)...))$
   c. Types and locks
4. MULTICS ring mechanism
   a. MULTICS rings: used for both data and procedures; rights are REWA
   b. (b_1, b_2) access bracket - can access freely; (b_3, b_4) call bracket - can call segment through gate; so if a's access bracket is (32,35) and its call bracket is (36,39), then *assuming permission mode (REWA) allows access*, a procedure in:
   rings 0-31: can access a, but ring-crossing fault occurs

rings 32-35: can access *a*, no ring-crossing fault
rings 36-39: can access *a*, provided a valid gate is used as an entry point
rings 40-63: cannot access *a*

c. If the procedure is accessing a data segment *d*, no call bracket allowed; given the above, *assuming permission mode (REWA) allows access*, a procedure in:
rings 0-32: can access *d*
rings 33-35: can access *d*, but cannot write to it (W or A)
rings 36-63: cannot access *d*

5. Assurance
    a. Trustworthy entities
    b. Security assurance
    c. Trusted system
    d. Why assurance is needed
    e. Requirements
    f. Assurance and software life cycle
6. Building trusted systems
    a. Stage 1: conception
    b. Stage 2: manufacture
    c. Stage 3: deployment
    d. Stage 4: maintenance
7. Life cycle: Waterfall Model
    a. Requirements definition and analysis
    b. System and software design (system design, program design)
    c. Implementation and unit testing
    d. Integration and system testing
    e. Operation and maintenance
8. Other life cycle models
    a. Exploratory programming
    b. Prototyping
    c. Formal transformation
    d. System assembly from reusable components
    e. Extreme programming