Using the trees to find the forest

Trustworthy computing as a systems-level issue

John James
John-James@usma.edu

Frank Mabry
Frank.Mabry@usma.edu
Trustworthy Computing Efforts

- NASA-Langley Goals (Chuck Meissner mid-80s):
  - Provably correct computer
  - Provably correct algorithms

- DARPA DSSA Program (Erik Mettala/Barry Boehm’90-’95):
  - Enable reuse through composition of components
  - Domain-specific models and architectures
  - Architectural Definition Languages (ADL)
  - Interface Definition Languages (IDL)

- NIST ATP on Component-Based Software (Barbara Cuthill‘95-’99)
  - Foster an industry based on composition of components
  - Productization of components

- DARPA project on Trustworthy Computing (Anup Ghosh’03)
  - Creating and maintaining trustworthy components
  - Related to reliability and software safety issues
Trustworthy Computing Challenges

• We build more complex systems than we can understand
  – We do not have accurate domain models for many systems
  – We do not know the failure modes of many existing systems

• We “move the aiming point” of existing systems:
  – Verification – “Am I building the system right?”
  – Validation – “Am I building the right system?”
  – Accreditation – “Do I trust that the system as built meets the requirements as stated?”

• We are focused on the trustworthiness of components
  – “Denial of service” attacks do not affect enterprise processes
  – Compromise of individual system components does not affect enterprise processes
Open-Loop and Closed-Loop Systems

Open-Loop (Model-View-Controller/Publish-And-Subscribe)

Closed-Loop (components are *not* independent)
One Approach: Intermediate levels of abstraction

- No ‘silver bullets” are available for solving the problem of trusting that the systems we build are working properly
- Focus on the system *purpose* (i.e. enable some enterprise process(es))
- System partitioning and evaluations help:
  - Maintain domain models
  - Use intermediate levels of aggregation
  - Use multiple sources of evaluation:
    - Network monitoring (i.e. intrusion detection and reaction)
    - Signals intelligence (i.e. jamming or signals intercept)
    - Human intelligence (i.e. greatest threat is internal)
Back-up
## Attack Plans

<table>
<thead>
<tr>
<th>Attack manufacturing</th>
<th>Attack Control</th>
<th>Attack Transportation</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Buffer overflow exploit</td>
<td>• Insider access from gambling debts</td>
<td>• Buffer overflow exploit</td>
</tr>
<tr>
<td>• Root kit for admin privileges</td>
<td>• Insider provides root access</td>
<td>• Root kit for admin privileges</td>
</tr>
<tr>
<td>• Expand to other network components</td>
<td>• Expand to other network components</td>
<td>• Expand to other network components</td>
</tr>
<tr>
<td>• Modify part orders</td>
<td>• Modify control plans</td>
<td>• Modify delivery schedules</td>
</tr>
<tr>
<td>• Modify shipping times</td>
<td>• Modify subordinate unit tasks</td>
<td>• Modify available units</td>
</tr>
<tr>
<td>• Modify delivery locations</td>
<td>• Modify modify subordinate unit commanded locations</td>
<td>• Modify indicated truck capacities</td>
</tr>
<tr>
<td>• Modify manufacturing tolerances</td>
<td>• Modify status reports to higher headquarters</td>
<td>• Modify pick up locations</td>
</tr>
<tr>
<td>• Modify interface requirements</td>
<td>• Modify timing of operations</td>
<td>• Modify delivery locations</td>
</tr>
<tr>
<td>• Indicate errors arise from faulty received parts</td>
<td>• Indicate errors arise from faulty orders</td>
<td>• Indicate errors arise from faulty requests</td>
</tr>
</tbody>
</table>
Some Environments for complex systems

*Architecture Description Language (ADL)*
*Interface Definition Language (IDL)*

- Mentor Graphics: VHDL-AMS as an ADL
- Mathworks: Matlab-Simulink (Hybrid state as an ADL)
- U.C. Berkeley: Ptolemy (Hybrid state as an ADL)
Real Time Support

- DARPA Open Control Platform (OCP) – based on a lightweight ORB
- Artisan – Real time UML (finite state only)