Trustworthiness

*Trustworthiness* is a measure of the extent to which a system, network, person, or other entity is worthy of being trusted to satisfy desired requirements (e.g., for security, reliability, human safety, system survivability despite realistic adversities, interoperability, ...), under certain assumptions.

Untrustworthiness

*Untrustworthiness* often results, due to incomplete requirements (specified vs desired), unsound architectures, bad implementation, dependence on untrustworthy 3rd parties, insider misuse, human frailty, invalid assumptions, sloppy evaluations, faulty risk analysis, etc.

Trust vs Trustworthiness

*Trust* is a slippery slope.

You may trust some entity because:
- You believe it is trustworthy.
- You don’t even know it exists.
- You are naïve or gullible.
- You ignore dialog boxes.
- You don’t check certificates.
- You know no alternatives.
Trustworthy Layered Architectures

- Multics 1965, PSOS, SIFT 1973
- MLS: KSOS, KVM, SeaView ...
- Composability, e.g., www.csl.sri.com/neumann/chats4.pdf

Layered Trustworthiness Compromises

Compromise results from misuse, HW faults, system failures, acts of God, ..., with three basic types

- Compromise from above/outside: penetrations, denials of service, untrustworthy 3rd parties, clouds, human frailty
- Compromise from within: insiders
- Compromise from below: subversion (overt or otherwise)

More Architectural Issues

- Trustworthy critical components
- Sound bases for composition
- Trusted bootload, trusted paths
- Cryptographic authentication
- Finer-grained authorization
- Traceback abilities
- Trustworthy code distribution
- Don’t forget denials of service
- Alternative hardware bases

Multics Hardware, 1965

- Virtual memory segments/pages, process isolation, address spaces, argument validation
- Access control interpretation
- Descriptor cache for performance
- Rings generalize supervisor/user to linearly-ordered domains, limit compromise from above.
Original Multics Software, 1965-1970

- Modular encapsulation, reentrant PL/I code; per-user processes;
dynamically linked symbolic file & names with access controls, I/O;
unified design philosophy – e.g.,
dependence on symbolic addressing,
dynamic segment linkage, paging,
stream I/O, command standards,
conventions, canonicalization, ...

MLS Multics: AIM, 1972
(Access Isolation Mechanism)

- Ring 1 MLS AIM provided 8 levels & 18 categories with very little
  performance degradation as Standard Multics feature (but seldom used).
  (8 MLS levels unrelated to 8 rings.)
  Secure audit logged failed accesses.

PSOS Design (SRI-NSA, 1973-80)

- Pervasive capability addressing,
tagged, typed, nonforgeable in HW/SW, nonbypassable
- Hierarchical encapsulated modular abstraction, object-oriented typing, extensible,
formally specified in SRI’s Hierarchical Development Methodology (HDM)

PSOS Capabilities

- Only two operations create capabilities: create new one, or create restricted copy.
- All objects are capability addressed, nonbypassably.
- Incremental trustworthiness. Capabilities accessible unless hidden by some layer.
- Capabilities could be tagged as propagation limited, MLS/MLI
Layer PSOS Abstraction or Functions

- 17* applications and user code (-)
- 16 user request interpreter *
- 15 user environments and name spaces *
- 14 user input-output *
- 13 procedure records *
- 12 user processes*, visible input-output*
- 11 creation and deletion of user objects*
- 10 directories (*)[c11]
- 9 extended types (*)[c11]
- 8 segmentation (+)[c11]
- 7 paging [6]
- 6 system processes, input-output [12]
- 5 primitive input/output [6]
- 4 arithmetic, other basic operations *
- 3 clocks [6]
- 2 interrupts [6]
- 1 registers (*), addressable memory [7]
- 0 capabilities * [MLS?]

* user-visible interface
(-) user-restrictable as desired
[c11] creation/deletion hidden by layer 11
[i] module hidden by layer i=6,7,8, or 12

---

PSOS Principled Assurance

- Pervasive assurance throughout cycles of development and use
- Assured multilevel security, with several possible alternative implementations.

---

PSOS Implementability

- Many lower-layer ops * would be directly executable from above, although some were hidden [ ].
- Multilevel security (MLS) could be embedded in layer 0 or as a secure object type.
- Hardware easily retrofittable.
- PSOS-like typing is used in Honeywell/SCC secure systems LoCK, SAT, SideWinder

---

SIFT (SRI/NASA)

Prototype ran nonstop at NASA Langley for many years.
SeaView (SRI/ONR-AFRL)

- SeaView, 1980s
- Application layer
- MLS-untrusted Oracle
- MLS kernel
  (Composition: ‘balanced assurance’)

Formal Methodology: HDM/PVS

- Use of HDM facilitated formal specification of modules, interlayer state mappings, and abstract implementations, enabling bottom-to-top analysis.
- A complex design became conceptually simple through its hierarchy and composable abstraction and encapsulation.
- Cf. PVS ‘interpretations’

KSOS (Ford Aerospace)

- 34-function MLS kernel, formally specified in HDM; Feiertag MLS flow analyzer used Boyer-Moore to find security flaws in specifications, most of which were fixed (except for a few detected covert storage channels). Also, code-to-spec consistency proof feasibility.

KVM Layers (SDC)

- MLS retrofit of kernel into IBM 370; NonKernel Control Program (one per security level); untrusted unmodified virtual operating system instances of MVS/MVT and VM/370 at desired MLS levels; users. Formal top- and 2nd-level specs formally verified with mappings in InaJo/FDM. Covert channel analysis.
What Is Needed More Generally?

- Holistic analysis must address sustainable democracy, social equality, and environments; also proactive attention to aging infrastructures such as bridges, roads, levees, railroads, flood control, healthcare; perhaps most important: holistic education.

Table A-1. TABLE OF INTERDEPENDENCIES
[Source: Roadmap for Cybersecurity Research, 2009]

<table>
<thead>
<tr>
<th>X</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalable Trustworthiness</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>Enterprise Metrics</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>Eval Methodology</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>Coping w Insiders</td>
<td>H</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>Coping w Malware</td>
<td>H</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>Global ID Mgt</td>
<td>H</td>
<td>M</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>Sys Survivability</td>
<td>H</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>L</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>Situ-Attribution</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>H</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>Provenance</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>H</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>Privacy-Aware Sec</td>
<td>M</td>
<td>M</td>
<td>H</td>
<td>L</td>
<td>M</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Usable Security</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>0</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>COLUMN TOTALS</th>
<th>H</th>
<th>5</th>
<th>1</th>
<th>2</th>
<th>7</th>
<th>4</th>
<th>5</th>
<th>8</th>
<th>4</th>
<th>4</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>5</td>
<td>9</td>
<td>7</td>
<td>3</td>
<td>2</td>
<td>6</td>
<td>5</td>
<td>2</td>
<td>6</td>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td>L</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

KVM References


Holistic System Approaches

- We need principled trustworthy systems with sound requirements, structured architectures, and proactive design for usability, evolvability, pervasive assurance; selective use of formal methods.
Plutarchies

Plutarch’s Greek writings stimulated among Romans considerable sense of the importance of understanding historical people and events. He observed that little seemed to have changed in human nature. Similarly, little has changed in commercial high-assurance systems, despite some major research advances. We need a better sense of history.

Saltzer-Schroeder-Kaashoek (1975+)

- Economy of mechanism
- Fail-safe defaults
- Complete mediation
- Open design
- Minimization of secrets *
- Separation of privilege
- Least privilege
- Least common mechanism
- Psychological acceptability
- Work factor
- Monitoring/auditing

Saltzer-Schroeder-Kaashoek

- J.H. Saltzer & M.D. Schroeder
- J.H. Saltzer & F. Kaashoek,

Some Computer-Related Conclusions

- We need proactive attention for computer-based infrastructures: trustworthy architectures, transparency, accountability...
- 20-20 foresight is much better than 20-20 hindsight. Priorities must be realistic and far-sighted.
Lessons Still To Be Learned

- Reliance on misapplied technology usually increases risks.
- With appropriate HW and system architecture, layered designs need not be inefficient.
- Eternal vigilance is required. (John Dewey: Each generation has to learn the lessons of the past all over again.)

Technological Desires

- Better system architectures
- Better system engineering
- Better public-private cooperation
- Better technology in education
- Practical privacy-aware crypto
- Nonproprietary systems: open source/arch/doc/composability
- (Leads to permanent job security?)

Possible Forcing Functions?

- Market forces are inadequate.
- Incentives for open systems, open interfaces, open source?
- Stronger regulation & liability?
- Tax incentives?
- Better awareness of the risks of untrustworthiness; disasters?
- Maybe some or all of the above?
- But there are no easy answers.

Overarching Lessons

- Don’t overendow technology.
- Every would-be technological solution has some risks, and escalates the attacks.
- Attackers have many advantages over defenders. However, too often systems collapse on their own without provocation.
PSOS 1980 Report


1979 PSOS Paper


2003 PSOS Revisited


PGN References

- Reflections on System Trustworthiness, Advances in Computing v.70, 2007:
A Few More PGN References

- The Role of Motherhood in the Pop Art of System Programming, ACM 2nd SOSP, 1969: multicians.org/pgn-motherhood.htm
- Computer-Related Risks, Addison-Wesley, 1995
- www.CSL.sri.com/neumann

<table>
<thead>
<tr>
<th>Group</th>
<th>PSOS Abstraction</th>
<th>layers</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>user/application activities</td>
<td>17–...</td>
</tr>
<tr>
<td>F</td>
<td>user abstractions</td>
<td>14–16</td>
</tr>
<tr>
<td>E</td>
<td>community abstractions</td>
<td>10–13</td>
</tr>
<tr>
<td>D</td>
<td>abstract object manager</td>
<td>9</td>
</tr>
<tr>
<td>C</td>
<td>virtual resources</td>
<td>6–8</td>
</tr>
<tr>
<td>B</td>
<td>physical resources</td>
<td>1–5</td>
</tr>
<tr>
<td>A</td>
<td>capabilities</td>
<td>0</td>
</tr>
</tbody>
</table>

layer | PSOS Abstraction or Function
17+   | applications and user code (-)
16    | user request interpreter (*)
15    | user environments and name spaces (*)
14    | user input-output (*)
13    | procedure records (*)
12    | user processes*, visible input-output*
11    | creation and deletion of user objects*
10    | directories (*)[c11]
9     | extended types (*)[c11]
8     | segmentation (*)[c11]
7     | paging [8]
6     | system processes, input-output [12]
5     | primitive input/output [6]
4     | arithmetic, other basic operations *
3     | clocks [6]
2     | interrupts [6]
1     | registers (*), addressable memory [7]
0     | capabilities *

Note:
* user-visible interface
* partially visible interface
* user-restrictable as desired
[c11] creation/deletion hidden by layer 11
[i] module hidden by layer i=6,7,8, or 12

Ideally all in hardware.
<table>
<thead>
<tr>
<th>layer</th>
<th>PSOS Abstraction or Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>17+</td>
<td>applications and user code (-)</td>
</tr>
<tr>
<td>16</td>
<td>user request interpreter *</td>
</tr>
<tr>
<td>15</td>
<td>user environments and name spaces *</td>
</tr>
<tr>
<td>14</td>
<td>user input-output *</td>
</tr>
<tr>
<td>13</td>
<td>procedure records *</td>
</tr>
<tr>
<td>12</td>
<td>user processes*, visible input-output*</td>
</tr>
<tr>
<td>11</td>
<td>creation and deletion of user objects*</td>
</tr>
<tr>
<td>10</td>
<td>directories (*)[c11]</td>
</tr>
<tr>
<td>9</td>
<td>extended types (*)[c11]</td>
</tr>
<tr>
<td>8</td>
<td>segmentation (*)[c11]</td>
</tr>
<tr>
<td>4</td>
<td>arithmetic, other basic operations *</td>
</tr>
<tr>
<td>1</td>
<td>registers (*)</td>
</tr>
<tr>
<td>0</td>
<td>capabilities *</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>layer</th>
<th>Properties for Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>17+</td>
<td>Application-relevant properties</td>
</tr>
<tr>
<td>16</td>
<td>Soundness of user types</td>
</tr>
<tr>
<td>15</td>
<td>Search-path flaw avoidance</td>
</tr>
<tr>
<td>12</td>
<td>Process isolation, no residues,</td>
</tr>
<tr>
<td>11</td>
<td>No lost objects</td>
</tr>
<tr>
<td>9</td>
<td>Generic type soundness</td>
</tr>
<tr>
<td>8</td>
<td>Segmentation integrity</td>
</tr>
<tr>
<td>6</td>
<td>Interrupts properly masked</td>
</tr>
<tr>
<td>4</td>
<td>Correctness of basic operations</td>
</tr>
<tr>
<td>0</td>
<td>Nonforgeable, nonbypassable, nonalterable capabilities</td>
</tr>
<tr>
<td></td>
<td>MLS (if present in layer 0)</td>
</tr>
</tbody>
</table>