

Managing Complexity with Principled Composability

Peter G. Neumann
Principal Scientist
SRI International ComputerSciLab
Menlo Park, CA 94025-3493
Neumann@CSL.sri.com
<http://www.csl.sri.com/neumann>
Tel 1-650-859-2375
Almaden Institute, April 2007

Complexity

- Simplicity is highly praised, but systems that must be trustworthy are inherently complex.
- Oversimplifying creates problems.
“Everything should be made as simple as possible, but no simpler.”
Albert Einstein
- Sound structure, composability, principles, discipline all can help.

Trustworthiness

A system (enterprise, method, crypto embedding, ...) is *trustworthy* with respect to specified requirements (security, reliability, human safety, interoperability, survivability despite realistic adversities and human weakness) if it satisfies those requirements with adequate assurance.

Problems

- System development failures
- Industry does not use what is known in the research community.

It cannot build small programs to work correctly.

It cannot build complex trustworthy complex systems.

It keeps repeating old mistakes.

Bad Development Practices

- Bad practices lead to monolithic nondecomposable/noncomposable hard-to-maintain systems, poor software engineering discipline, unsafe programming languages, overdependence on patches, short-term optimization, ... They create many problems.

Typical Software Flaws 1

- Unfortunately, common types of mistakes (design flaws, software bugs, operational errors) recur.
- Buffer overflows, unchecked limits, type mismatches, and invalid parameters are ubiquitous – which is extraordinarily short-sighted and preventable.

Typical Software Flaws 2

- Programming languages are easily misused, even good ones.
- Absence of development tools.
Useful static analysis tools include Coverity Prevent, Fortify, StackGuard, LibSafe, RaceTrack, Hao Chen's MOPS, slint, and many more.

Propagation Risks 1: Widespread Network Outages

- 1980 ARPANET collapse: router memory errors, weak garbage collection of old status messages, memory overflow in every node
- 1990 AT&T longlines collapse: untested change in recovery code, repeated crashing for half a day.

More Propagation Risks 2: Widespread power outages

- Northeast US, Nov 1965
- Lower NY State, Jul 1977, >26 hrs
- Ten Western states, Oct 1984
- Western US, Jul 1996, heat/tree
- Western US/Canada/Baja, Aug 1996
- Northeast, Aug 2003, >2 days
- Queens NY, week-long, aging wiring

Backup and Recovery Risks 1: Air-Traffic Control Failures

- LA Palmdale ATC Jul 2006 power
- Reagan National Apr 2000 power
- LI NY ATC SW upgrade Jun 1998
- LA ElToro ATC 104 failures/day
1989 (no previous system saved)
- 3 NY airports 1991 (on batteries)

Backup and Recovery Risks 2

More total system failures & backup:

- Swedish central train res system
- Washington Metro Blue Line 1997
- SF BART SW upgrades Apr 2006
- Japanese stock exchange Nov 2005

Cases of losses with no backup:

- NY Public library references
- Dutch criminal mgmt system

Paradigmatic Example: Voting 1

- Elections should have end-to-end integrity/reliability/accountability, nonsubvertible audit trails, uncompromised voter privacy, etc.
- The entire process is vulnerable: registration, voter authentication, authorization, voting, counting, certifying, recounting, etc.

Voting 2

- Weakness in depth: every step is a potential weak link.
- All-electronic paperless systems are unauditible, lacking integrity, and subject to undetectable errors, fraud, and many extrinsic nontechnological problems.
- HAVA, EAC, evals: simplistic

Voting 3

- Huge differences exist between what is known in research and what exists in development practice, federal standards, evaluation, ...
- Many problems are nontechnical (absentee ballots, politics, ...).

Principled System Development

- Holistic approaches to complexity: sound requirements, structured architectures, principles, good software engineering practice, predictable composability; proactive design for usability, evolvability, trustworthiness, and pervasive assurance; selective use of formal methods, and more.

Principled System Design

- Management of complexity through constructive architectures that enable facile composability and modularly encapsulate what must be trustworthy, such as separation kernels, virtualization, alternative approaches to multiple security levels, self-repairing systems, ...

Principled System Implementation

- Sound software engineering, sound programming languages
- Well-supported composability
- Property-preserving refinements
- Extensive software analysis
- End-to-end self-checking

Principled System Assurance

- Pervasive assurance throughout cycles of development and use
- Assured composability, with hierarchical closure as in PSOS (see Robinson-Levitt 1977)
- Assured separation, controlled sharing, multilevel security (!)

Saltzer-Schroeder Principles (1975)

- Economy of mechanism
- Fail-safe defaults
- Complete mediation
- Open design
- Separation of privilege
- Least privilege
- Least common mechanism
- Psychological acceptability
- Work factor
- Recording of compromises

Composability Principles (PGN, 2004)

Sound distributed hierarchical architectures; minimization of what must be trusted; abstraction, encapsulation, robust dependency; separation of policy-mechanism, roles, domains; sound authentication, authorization, access control; administrative controllability, comprehensive accountability

Principled Composable Architectures

- Realistic application of principles
- Architectural trustworthiness and principled proactive composability
- Composability of requirements, specs, systems, models, analyses
- Perspicuous composable interfaces
- Making composability practical:
www.csl.sri.com/neumann/chats4.^{*}
(your choice of html, pdf, ps)

Various Architectural Structures

- Architectural trustworthiness,
e.g., Multics, PSOS, VMMs,
Rushby isolation kernel 1982,
Rushby-Randell DSS 1983,
Proctor-Neumann 1992,
www.csl.sri.com/neumann/ncs92.html
Rushby-DeLong 2007
- Trustworthiness enhancement
www.csl.sri.com/neumann/chats4.*

Old Examples of Principled Systems

- Multics: Rings, symbolic naming of files, I/O, EPL, no stack overflows (and no Y2K!), discipline
- PSOS: Nonforgeable capabilities, abstraction, encapsulation, strong typing, efficient hierarchy, formal specs/proofs
- SeaView: MLS DBMS with no MLS trust needed in Oracle

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?

Conclusions

- Complexity can be managed with principled composability of requirements, architecture, implementation, policies, ...
- Good support tools and early analysis yield higher assurance.
- Foresight has enormous payoffs.
- Trust only what is trustworthy.
- People are critical resources: education, research, vision.

A Few Relevant References

- PGN, Reflections on System Trustworthiness, in *Advances in Computers 70*, Elsevier, 2007
- Principled Assuredly Trustworthy Composable Architectures:
www.CSL.sri.com/neumann/chats4.html, .pdf, .ps
- PSOS Revisited, ACSAC 2003:
www.csl.sri.com/neumann/psos03.pdf
- ACM Risks Forum, www.risks.org
- PGN, www.CSL.sri.com/neumann

Old PSOS/HDM References

-
- L. Robinson, K.N. Levitt, Proof Techniques for Hierarchically Structured Programs, *CACM*, Apr 1977.
 - Neumann, Boyer, Feiertag, Levitt, Robinson, A Provably Secure Operating System: The System, Its Applications, and Proofs, SRI 1980
<http://www.csl.sri.com/neumann/psos/psos80.ps>