A Personal History of Layered Trustworthiness

Peter G. Neumann
SRI International
Computer Science Laboratory
Menlo Park CA 94025-3493
Neumann@csl.sri.com
1-650-859-2375

These slides were used in the Elliott Organick Memorial Lectures University of Utah, 26-27 March 2013 along with project slides: see http://www.csl.sri.com/neumann

These slides are approved for public release, distribution unlimited. Some of the material in this talk is based on a project supported by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL under contract FA8750-10-C-0237. The views expressed are mine and do not reflect the official policy or position of the Department of of Defense or the U.S. Government.

Trustworthiness

Trustworthiness is a multidimensional measure of the extent to which an entity (person, system, network, etc.) is worthy of being trusted to satisfy certain requirements for system security, system integrity, system reliability, human safety, system survivability despite realistic adversities, real-time performance, interoperability, ... at particular layers of abstraction.

Untrustworthiness

Untrustworthiness (e.g., failure to satisfy critical needs) is ubiquitous, due to incomplete requirements (specified vs desired), unsound architectures, bad implementation, dependence on untrustworthy 3rd parties, insider misuse [8], human frailty, invalid assumptions, sloppy evaluations, faulty risk analysis, etc.

- Weakness in depth *and* breadth!

Unsound Architectures

- Building elaborate castles in sand
- Poor modularization (e.g., lack of encapsulation/information hiding)
- Poor hierarchical abstraction (e.g., flat software, without layered protection)
- Anti-hierarchical dependencies (e.g., circular; dependence on less trustworthy components [20])
- Many other causes

Emergent Properties

System Security, Survivability & Human Safety are multidimensional emergent properties of systems and networks, requiring trustworthy foundations, reliability, robustness, resilience, interoperability, compatibility, predictability, usability, sufficiently trustworthy people, & much more.

Trustworthiness Interdependencies

- Unreliable systems are not secure.
- Insecure systems are not reliable.
- Human safety and total-system survivability require security, reliability, timeliness, and more.
- Trustworthiness requirements may interact with one another, with bad emergent properties,
- People add significant risks.
- Compromises can be pervasive.

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, people
- Compromise from within: insider misuse, software flaws
- 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
- Tamper-resistant accountability
- Don't forget denials of service
- Trustworthy code distribution
- Lots more to think about ...

Trust

Trust is what you have to do –
whether you like it or not –
when you must depend on something
to satisfy desired properties.

- Today we have *faith-based trust.*

We need significant trustworthiness!

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.

Trust is a Belief Relation

Mathematically/logically,

`W trusts X to satisfy Y and Z.`

W may be person(s) and/or agent(s).

X may be a collection of systems,
information, people, and so on.

Y might be a set of expectations.

Z might be mitigating constraints.

(Cf. Burrows, Abadi, Needham,
Belief Logic [28].)

A Basic Goal for Trust

- We have learned much about achieving trustworthiness in systems, but not enough.
- We need corresponding ways to systematize networked trust in persons, artifacts, information, communications, content, etc., and to advance trustworthiness.

Built-in System Integrity

- Intuitively, beware of dependence on less trustworthy entities. This is the essence of Biba [20].
- In practice, we can sometimes overcome such dependencies, e.g., error-correcting codes, fault-tolerant computing, cryptography, dynamic checks, and other mechanisms. See [4].

Trustworthy Systems

- For meaningfully trustworthy critical systems, incremental changes are likely to be fundamentally inadequate: You cannot get there from here.
- We consider some hierarchical system architectures, some of which were mostly incremental, others much more clean slate.

Trustworthy Layered Architectures

- Multics 1965, PSOS, SIFT 1973
- Dijkstra, THE 1968 [27]
- Rushby separation kernel 1982 to Rushby-DeLong 2007–
- MLS: KSOS, KVM, SeaView ...
- MILS: Rushby-Randell DSS 1983, NSA 1988, SRI 1992 [14]
- Virtualization: KVM 1977, Rushby-DeLong 2007–
- Capsicum [21], CHERI [22,23]
- Composability [4]

Relevant Mentors/Inspirers

- Albert Einstein 1952
- Multics: Corbató, Glaser, Daley, Saltzer, Osanna, Vyssotsky, McIlroy, Bob Morris, Ken Thompson, et al.
- E.W. Dijkstra 1968
- PSOS: Boyer, Feiertag, Levitt, Robinson, 1973
- Dave Parnas, Brian Randell, Jim Horning, Marv Schaefer, ... 1970s
- Robert N.M. Watson, Doug Maughan, Howie Shrobe, 2000–

Multics Hardware, 1965 [29]

- Independent virtual memory segments, paging, address spaces; process isolation
- Access control interpretation
- Descriptor cache for performance
- Hierarchical integrity: 8 rings extend supervisor/user modes, avoid compromise from above – layer by layer.

Multics Software, 1965-1970 [29]

Modular encapsulation, reentrant PL/I code; per-user processes; Directory hierarchy w. ACLs & dynamic linking of symbolic file and I/O names; ring X-ing; unified design philosophy – e.g., dependence on symbolic addressing, stream I/O, command standards, conventions, argument validation iexceptions, canonicalization, ... pull-only version control, search ordering, script-based commands

MLS Multics: AIM, 1972 (Access Isolation Mechanism)

Ring 1 MLS AIM: 8 security levels & 18 categories with very little performance degradation as Standard Multics feature (not often 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 user-definable types, HW/SW formally specified in SRI's Hierarchical Development Methodology (HDM) [1-3]

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 <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 *  [could include MLS]
---------------------------------------
  *   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
```



```
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 <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 *  [could include MLS]
```

PSOS Principled Assurance

- Pervasive assurance throughout cycles of development and use
- Assured composability, layered hierarchical noncompromisibility (Robinson-Levitt 1977 [15]). Cf. the UTexas CLInc stack.
- Assured multilevel security, with several possible alternative implementations.

PSOS Implementability

- Many lower-layer ops '*' would be directly executable from above, others could be hidden '[ ]'.
- Multilevel security (MLS) could be embedded in layer 0, or in a primitive secure object type.
- Hardware easily retrofittable.
- Honeywell/SCC secure systems LoCK, SAT, SideWinder adopted PSOS's use of type safety.

SRI's SIFT (aka NASA's SwIFT)

• SIFT: Software Implemented Fault Tolerance, 1973-2000:
  Application layer
  Voting layer (2 of 3)
  Broadcast layer
  Synchronization layer
  7x (Bendix avionics CPUs)
$p(failure) = 10^{-10}/hr$, 10^{-5} better than 1 CPU. Never crashed for 20+ years. Wensley/Levitt/Green/Goldberg/PGN 1973, plus many later papers.

KSOS (Ford Aerospace, 1980s)

34-function MLS kernel, formally specified in HDM; SRI used the Feiertag MLS flow analyzer and Boyer-Moore theorem prover 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, mid-1970s)

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. [16,17]

SeaView (SRI/ONR-AFRL)

- Multilevel Secure DBMS, 1980s:
  Application layer
  MLS-untrusted Oracle
  An evaluated MLS kernel

Composition, 'balanced assurance': System at evaluated kernel rating despite Oracle not MLS, after Oracle removed shared buffers.

DARPA CRASH Program

- CRASH = Clean-slate design of Resilient, Adaptive, Secure Hosts:

Howard Shrobe is the DARPA PM.

- Our SRI/U.Cambridge CRASH project is called CTSRD: CRASH-worthy Trustworthy Systems R&D.

CTSRD = CRASH-worthy Trustworthy Systems R&D

We're building a principled, formally supported, robust hardware/software platform designed for technology transfer. Security design principles and program security structure are reinforced by Temporally Enforced Security Logic Assertions (TESLA) and Capability Hardware Enhanced RISC Instructions (CHERI [22,23]).

Historical Basis for CTSRD

CTSRD is inspired by the formally specified tagged/typed HW/SW hierarchical capability-based design of SRI's PSOS [1,2,3], Rushby's separation kernels and recent work with DeLong, Watson's Capsicum [21], FreeBSD, & Chrome OS, and related efforts. The hybrid architecture and formal methods lead to significant advances.

Systemic Considerations

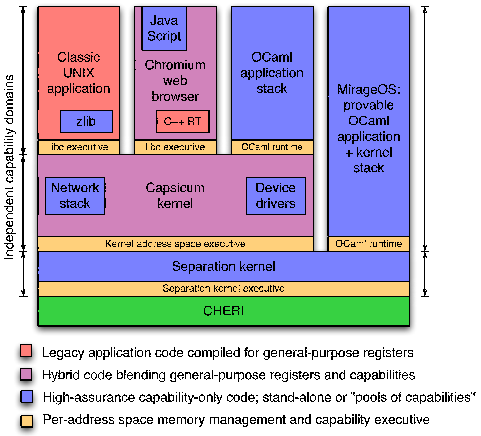
- Dramatic changes in threats
- New research opportunities
- HL programmable FPGAs
- Exposing hardware parallelism
- Advances in programming languages
- New HW/OS open-source variants (MIPS,ARM; iOS/Android)
- Advances in open source SW
- Advances in formal methods

Key Tenets of CTSRD

- Typed-object program-aware HW simplifies total assurance
- Fine-grained compartmentalizing of applications without sacrificing performance/programmability of paged VM architectures.
- Secure division of memory within address spaces; message passing handled by compilers

CTSRD Architecture 1 [22,23]

- CTSRD's architecture is a hybrid design, with high-assurance code for CHERI and TESLA coexisting with legacy OSs and software, with a gradual adoption path for high-assurance security features.
- Code may be compiled for general-purpose and/or capability registers.
- Each address space has an executive for memory allocation and capability semantics (creation etc.).



CTSRD Architecture 2

- Thread contexts may be limited to using only capability addressing.
- 'Capability pools' let capability code operate within hybrid processes (Capsicum kernel, libraries, script interpreters, etc.).
- High-assurance components (such as a separation kernel and MirageOS stack) would use capabilities and be compiled with TESLA assertions to detect violations of principles.

CTSRD Architecture 3

- CTSRD supports critical TCB components: separation kernels, language runtimes, and particularly exposed or frequently vulnerable software components.
- Formal analysis can give confidence in the design and implementation. TESLA picks up at runtime where formal analysis leaves off.

CHERI 1

- The CHERI hardware architecture is specified in Bluespec and implemented in an FPGA soft core: capability instructions and paged virtual memory in HW.
- Program security structure is exposed by the compiler to (and enforced by) hardware: general-purpose RISC registers + capability registers and tagged memory. (Multics-like rings are implicit.)

CHERI 2

- Bluespec restructured for ease of formal analysis. SRI FM tools are inside the development tool chain.
- Massive multi-threading implements procedure capabilities with hardware message passing (vs. expensive virtual-memory context switches).
- Multicore is also implemented.

CHERI Development Platform

Our MIPS64 hardware began with Cambridge's TIGER MIPS64 soft core with added capability instructions/ and registers. It is specified in Bluespec (high-level, abstract, typed); BSV compiles specs into Verilog for Terasic DE4 Altera FPGA and Xilinx NetFPGA boards.

Examples of HW-SW Co-design

- PSOS HW/SW in SPECIAL [2,3]: SPECIfication & Assertion Language state-machine based, nonexecutable
- Elliott Organick: Serious efforts on HW/SW abstract co-design: "Ada to Silicon" [30]
- Nirav Dave PhD Thesis: Bluespec BSV extended to BSL with movable HW/SW boundary: executable, compiles into Verilog or desired programming languages [24].

Bottom-to-Top Layered Assurance

- Consistently capability-aware architecture mirrors hierarchical incremental assurance (slide 30): HW/kernel/compiler/apps.
- Layered selective use of formal methods from the hardware up: SRI's PVS, SAL model checking, YICES SMT solver, other tools applied to HW/SW/compilers.

DARPA MRC Program

- MRC = Mission-oriented Resilient Clouds:

Howard Shrobe is the DARPA PM.

- Our SRI/U.Cambridge MRC project is called $(MRC)^2$: Modular Research-based Composably trustworthy MRC.
- Trustworthy Software-Defined Networking & Datacenters, Switch/controller co-design.
- More on CTSRD, $(MRC)^2$ follows.

Simplicity: Fundamental Quote

"Everything should be made as simple as possible, *but no simpler.*"
Albert Einstein

Violations of that principle are responsible for many breakdowns in trust and trustworthiness, e.g., poor usability (too complex), unforeseen risks (too simple).

Simplicity and Complexity

- *Simplicity* is highly praised, but oversimplifying almost always creates problems.
- Attaining trustworthiness is *inherently complex* even with highly trustworthy components, much harder in the presence of anything that is untrustworthy.

Simplifying Inherent Complexity

Layered & distributed architectures, vertical and horizontal abstraction with encapsulation, virtualized interfaces, invisible encryption, and more can help mask underlying complexity, rigorize system-wide analysis, and yield operationally effective trustworthiness.

Trustworthy Composition [4]

- *Composition* is meaningful at many layers of abstraction with respect to policies, protocols, specs, components, program language features, proofs, evaluations, ...
- Vertical, horizontal, parallel, sequential, distributed, refinement, X-tolerant, ...
- Amplifying trustworthiness [4]

Composability/Compositionality

- Predictable Composition is crucial to managing complexity.

www.csl.sri.com/neumann/chats4.pdf

- Composability: Are properties preserved or soundly transformed? critical system requirements?
- Compositionality: Are emergent properties consistent with critical system requirements?

Impediments to Composition

- Incompatible definitions
- Inadequate architectures
- Incompatible interfaces
- Nonencapsulated modularity
- Hidden state interactions, memory residues, leaks, etc.
- Undocumented side-effects
- Unanalyzed emergent properties
- Undisciplined software practices
- Unsafe programming languages

The So-Called Perimeter of Trust

This is also a multidimensional slippery slope! There is typically *no single perimeter,* other than the totality of every user on every accessible system (the Internet?). Each property may have its own would-be (fuzzy) perimeter. Insiders vs Outsiders? Also fuzzy: Clouds can be virtual insiders!

Illustrative Example: Elections

Elections represent a paradigmatic hard problem in which insiders and outsiders have huge opportunities for misuse, often with minimal oversight. Untrustworthy third parties abound inside the supply chain and the election process. Beginning-to-end and bottom-to-top assurance are completely absent, but urgently needed.

Insider Fraud in Elections

- Clay Cty, Kentucky: 8 convictions for insider fraud (rigging, contract bribes, 2002, 2004, 2006), including a circuit court judge.
- Ohio: 2 indictments for election fraud; election managers mishandled ballots, leaked partial tallies; illegal cables; clock alterations; log gaps; shared account; no dual control. 2006

More Insider Fraud in Elections

- Indiana: 7 felony indictments
- Colorado: current Grand Jury
- Many others suspected

(weak authentication, accountability, oversight; equipment sleep-overs, etc.)

- Cf. Nevada gambling frauds: Strong regulations, weak oversight. Gaming Board member embezzled, convicted, murdered. See Jeff Burbank: EVT/WOTE 2010 video; *License to Steal*

A Realistic View of Networks

"To a first approximation, every computer in the world is connected with every other computer."
Bob Morris, then Chief Scientist, National Computer Security Center, briefing the CSTB, 9/19/1988.
NSA Chief Sci. K. Speierman and PGN also warned of impending chaos.

A Holistic View of the Future

We cannot have a trustworthy networked Web environment unless we have secure/resilient systems, servers, switches, routers, switch controllers, and cryptographic embeddings – with operational stability, autonomous maintainability, and much more. This is a very difficult challenge.

Holistic Approaches

- We need principled trustworthy networked systems with sound requirements and architectures, proactive design for security, reliability, resilience, usability, evolvability, pervasive assurance, selective use of formal methods, predictably composable [4], holistic foresight [25,26].

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?
- Professional certification?
- Tax incentives?
- Better awareness of the risks of untrustworthiness; disasters?
- Maybe some or all of the above?
- But there are no easy answers.

Conclusions 1

- *Weakness in breadth and depth* must be avoided!
- We need constructive attention to high-assurance trustworthiness for national infrastructures: trustworthy architectures, transparency, accountability...
- We should not have to trust untrustworthy systems!

Conclusions 2

- We need long-term total-system life-cycle approaches, far-sighted optimization, and more [25,26].
- Above all, we need far-sighted research, development, commitment to proactive scientifically sound approaches, and education.
- Time may be ripe for CTSRD's formally based hybrid approach, with coordinated program-aware HW/SW/languages/compilers.

Hierarchies and 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.

References: PSOS 1980 Report

1. P.G. Neumann, R.S. Boyer, R.J. Feiertag, K.N. Levitt, L. Robinson, A Provably Secure Operating System: The System, Its Applications, and Proofs, SRI International, Computer Science Laboratory, 2nd edition, Report CSL-116, May 1980.
http://www.csl.sri.com/neumann/psos/psos80.pdf and .ps

1979 PSOS Paper

2. R.J. Feiertag, P.G. Neumann, The Foundations of a Provably Secure Operating System (PSOS), Proceedings of the National Computer Conference, AFIPS Press, 1979, 329–334.
http://www.csl.sri.com/neumann/psos.pdf

2003 PSOS Revisited

3. P.G. Neumann, R.J. Feiertag, PSOS Revisited, Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC 2003), Classic Papers section, IEEE Computer Society, Las Vegas NV, December 2003, 208–216.
http://www.csl.sri.com/neumann/psos03.pdf

More PGN References

4. Principled assuredly trustworthy composable architectures, 2004 (includes a wonderful appendix by Virgil Gligor on system modularity).
http://www.CSL.sri.com/neumann/chats4.html, .pdf, .ps
5. Reflections on System Trustworthiness, Advances in Computing v.70, Academic Press (Elsevier), 2007, 267–309.

More PGN References

6. Holistic Systems, *ACM SIGSOFT Softw.Eng.Notes,* Nov. 2006. http://www.csl.sri.com/neumann/holistic.pdf
7. Practical Architectures for Survivable Systems and Networks, ARL, June 2000. http://www.csl.sri.com/neumann/survivability.html neumann/survivability.pdf

More PGN References

8. PGN, Combatting Insider Threats, in *Insider Threats in Cyber Security and Beyond,* Christian Probst, Jeffrey Hunker, Dieter Gollmann, and Matt Bishop (editors), Springer Verlag, 2010.
9. The Role of Motherhood in the Pop Art of System Programming, SOSP, 1969. www.multicians.org/pgn-motherhood.html

More PGN References

10. Computer-Related Risks: Addison-Wesley, 1995.
11. www.CSL.sri.com/neumann/ (elections: /illustrative.html)
12. ACM Risks Forum, www.risks.org (every RISKS issue, since 1984)

Other Co-authored References 4

13. Highly Dependable Distributed Systems, PGN & Les Lamport, US Army CECOM, June 1993.
14. N.E. Proctor and PGN, Architectural Implications of Covert Channels, Proc. 15th National Computer Security Conf. 1992. http://www.csl.sri.com/neumann/ncs92.html

Robinson-Levitt 1977

15. Larry Robinson and Karl Levitt, Proof Techniques for Hierarchically Structured Programs, *Comm. ACM*, 20, 4, 271–283, April 1977.

KVM References

16. M. Schaefer, BD Gold, RR Linde, JF Scheid, Program confinement in KVM/370, Proc. 1977 ACM Annual Conference, Seattle, 404-410). (Flaw discovered in Amdahl HW!)

17. BD Gold, RR Linde, PF Cudney, KVM/370 in Retrospect, IEEE SSP, Oakland, 1984, 13–23.

Saltzer-Schroeder-Kaashoek

18. J.H. Saltzer & M.D. Schroeder The Protection of Information in Computer Systems, *Proc. IEEE 63*, 9, September 1975, 1278–1308. http://www.multicians.org

19. J.H. Saltzer & F. Kaashoek, Principles of Computer System Design, Morgan Kauffman, 2009, Chapters 1-6; 7-11 are online. http://ocw.mit.edu/Saltzer-Kaashoek

Biba Integrity

20. K.J. Biba, Integrity Considerations for Secure Computer Systems, The Mitre Corporation technical report MTR 3153, June 1975. http://seclab.cs.ucdavis.edu/projects/history/papers/biba75.pdf

Capsicum: Hybrid Architecture

21. Robert N.M. Watson, Jon Anderson, Ben Laurie, and Kris Kennaway, Capsicum: Practical Capabilities for Unix, *Proceedings of the 19th USENIX Security Symposium,* August 2010.

CTSRD System Architecture 1

22. Peter G. Neumann and Robert N.M. Watson (Univ. of Cambridge), Capabilities Revisited: A Holistic Approach to Bottom-to-Top Assurance of Trustworthy Systems, Layered Assurance Workshop, Austin TX, 6-7 December 2010 http://www.csl.sri.com/neumann/law10.pdf

CTSRD System Architecture 2

23. R.N.M. Watson, P.G. Neumann,... A Research Platform Deconflating Hardware Virtualization & Protection RESoLVE Workshop, ASPLOS, London, UK, March 2012 http://www.csl.sri.com/neumann/2012resolve-cheri.pdf

BSL HW/SW Co-design

24. Nirav Davé. A Unified Model for Hardware/Software Co-design, MIT, Cambridge, Mass., 2011. http://people.csail.mit.edu/ndave/

Inside Risks: Foresight

- 25. PGN, The Foresight Saga, Redux: Short-term thinking is the enemy of the long-term future, CACM, Oct 2012.
- 26. PGN, More Sight on Foresight: Reflecting on Elections, Natural Disasters, and the Future, CACM, Feb 2013.

Dijkstra THE System; BAN Logic

27. E.W. Dijkstra, The Structure of the THE Multiprogramming System, *Communications of the ACM, 11,* 5, May 1968, 341–346.

28. M. Burrows, M. Abadi, R. Needham, A Logic of Authentication, *ACM Trans. Computer Systems 8*, 1, Feb 1990, 18-36.

Other References

- 29. Elliott Organick, The Multics System: An Examination of Its Structure, MIT Press, 1972. (Elliott wrote 19 books!)
- 30. Elliott Organick, 1925-1985, *Comm. ACM* 29 3, March 1986, p. 231.
- See my website for many more.