System and Network Trustworthiness in Perspective

Peter G. Neumann
Computer Science Laboratory, Principled Systems Group
SRI International, Menlo Park CA 94025-3493

ABSTRACT

Characteristic problem areas experienced in the past are considered here, as well as some of the challenges that must be confronted in trying to achieve greater trustworthiness in computer systems and networks and in the overall environments in which they must operate. Some system development recommendations for the future are also discussed.

Category

D.4.6 Security and Protection

General Terms

Design, security, reliability, verification

Keywords

Computer systems, networks, trustworthiness, security, reliability, survivability, assurance, threats, vulnerabilities, risks

1. INTRODUCTION

In the context of this paper, *trustworthiness* means simply that something is worthy of being trusted to satisfy its specified requirements, typically relating to overall information system properties such as security, reliability, human safety, and survivability in the presence of a wide range of adversities, subject to some sort of assurance measures. In this context, *security* can be broadly thought of as encompassing everything that promotes the prevention, detection, and remediation of deleterious system behavior (for example, including actions of people, information technology, and physical environments).

Trustworthiness is relevant throughout system life cycles, from conception to requirements to detailed architectural designs to implementation, and then continuing into use, operation, maintenance, and recycling through revisions as appropriate. Trustworthiness is very difficult to achieve unless it is systematically integrated throughout the development cycle.

We begin with a few high-level remarks that might help motivate some of the problems that arise in designing, developing, operating, maintaining, and using complex highly distributed systems and networks — and especially those with extensive requirements for trustworthiness.

- "The future isn't what it used to be."[1] It is becoming ever more difficult to keep up with critical needs for trustworthy applications in the face of increasing system and network security vulnerabilities, increasing development complexity, multiple fault modes, sophisticated threats from insiders and outsiders, ubiquitously increasing societal dependence on information technology, increased dependence on the roles of administrators and users, and increasing costs and disruptions associated with the resulting risks. All these factors — and more — tend to exacerbate the problems of developing trustworthy intensively computer-based enterprises.

- "We need to go back to the future." Past research and development efforts present many important lessons for the future, although many of those lessons are either forgotten or never learned. Thus, it is useful to reconsider how we arrived at the present state of the art, and extrapolate on what might be needed in the future. We consider here both short-term and long-term approaches, and outline some of the hard problems that must be more systematically addressed.

- "Progress is always slower than you'd think." Proactive system development is often shunned, but has enormous potential payoffs. Mistakes of the past are frequently repeated, many of which could be easily surmounted — for example, using approaches such as evolvable system architectures, development that facilitates pervasive and predictable subsystem composability, intelligent interface design, sound system embeddings of strong cryptography, sound analysis tools, and up-front attention to assured trustworthiness.

2. PRINCIPLED SYSTEMS

For many years, the security community has been evolving various principles for the development of trustworthy systems, such as [14] and the Saltzer-Schroeder principles [28], both of which grew out of the Multics development.

An extended approach to principled development [16] also provides some analysis of how these principles and others can affect trustworthiness and assurance of systems as a whole. Several aspects of such an approach are highlighted here, notably the principle of least privilege; minimization of the extent to which portions of a system or network must be trusted, thereby limiting what is

[1] I first heard this almost-Yogi-Berra-like statement in 1969 in a keynote address by Arthur Clarke, who lamented that it was becoming ever more difficult to write good science fiction.

CCS'06, October 30–November 3, 2006, Alexandria, Virginia, USA.

vulnerable to accidents and misuse; modular abstraction with encapsulation and complete mediation; and thorough nonbypassable auditing, In addition, various principles under the rubric of good software engineering practice can add significantly to assured trustworthiness, which — in addition to the aforementioned properties relating to security, reliability, safety, and survivability — can also encompass other concepts such as predictable composability, assured interoperability, and real-time performance.

Multics was perhaps the most principled of system developments. It broke some important new ground with hardware-software support for virtual memory, segmentation, paging, protection domains, stacks in which anything outside of the current stack frame was nonexecutable and nonaddressable, hierarchical directories, access control lists, dynamic linking, the use of a stark subset of PL/I that naturally prevented certain flaws, and so on. However, most instructive may have been its development discipline [5, 6].

Other principled efforts from the 1960s and 1970s include the T.H.E. system and its use of a deadlock-avoiding hierarchical locking strategy described by Dijkstra [7], and a seminal series of papers by Parnas (e.g., [20]). The 1977 Robinson–Levitt paper [22] on hierarchical formal specifications introduced the concept of formal mappings between different layers of functional specifications that represent abstract implementations of each layer as a function of the lower layers, as the foundation for the design of the Provably Secure Operating System (PSOS) [8, 17, 18].

The desire for multilevel security (MLS) prompted considerable research and development in the 1970s and 1980s, although those efforts did not bear much fruit with respect to commercially available MLS systems with strong assurance. Various efforts that were countercultural at the time (e.g., Rushby-Randell [27] and Proctor-Neumann [21]) are gaining new currency with respect to the concept of Multiple Independent Levels of Security (MILS) within the Secure Global Information Grid.

In addition, there is renewed interest in Rushby's 1981 notion of a separation kernel [26] and also in virtual machine monitors (such as VMware and Xen), which have a basic need for disciplined but limited interpartition communications [3].

Principles that are generally accepted *in principle* are often not widely used *in practice.* There are several possible explanations for this. For example, principles are not absolute; they are also not independent, and may sometimes interfere with one another; and their effective use requires appropriate educational stimuli, experiential feedback, acquired discipline, and (above all) foresight — which often runs counter to short-term motives such as profits and marketplace.

To illustrate the relevance of principled developments, we consider some particularly problematic real-world uses of computers and communications in the next two sections.

3. NETWORK DISRUPTIONS

Past networking difficulties offer lessons that need to be assimilated by researchers and system developers. This section revisits some old and new history relating to widespread network outages and other extreme network behavior. The earlier history may be of particular interest to younger researchers and developers who were not active in the olden days.

Various examples of widespread propagation effects are illustrative of some of the complexities arising in networked systems. Of particular interest are two cases in which total network failure modes resulted from local faults — namely the 1980 ARPANET collapse and the 1990 AT&T long-distance collapse — as well as numerous instances of domino-like propagating power outages.

- The 1980 ARPANET collapse: As a result of unchecked bits dropped in the highest-priority status messages in the memory of one ARPANET router (a so-called Interface Message Processor) and a garbage-collection algorithm that could subsequently not eliminate any nonrecent status messages, a memory overflow occurred in every node in the network. It took 4 hours to diagnose the unprecedented failure mode, telephone the administrator of every node and request that it be shut down manually, and finally, upon confirmation that all nodes had been shut down, manually initiate restarting every node. (See [24].)

- The 1990 AT&T long-lines collapse: An untested upgrade to the fault-tolerant recovery software had been installed in 114 ESS Number 4 telephone switches. (The C program contained a `break` statement within an `if` clause nested within a `switch` clause.) On 15 January 1990, one node crashed, and auto-recovered. However, as a result of the flaw, all the immediate neighbor nodes crashed, followed by iterated crashes of every node in the network repeatedly for something approaching half a day. Completing long-distance telephone calls became almost impossible. (See the ACM Risks Forum, vol 9, number 61, and *Telephony,* 22 January 1990, p. 11.)

In both of these cases, the developers of the networking technology seemed to have believed that network-wide outages could not possibly result from single-point failure modes.

Computer-controlled electrical-power networks have also been implicated in numerous widespread propagating power outages, evidently with some *a priori* disbelief on the part of developers and operators that widespread outages could result. (References for these and other cases noted below can be found in [13], with some of the older cases documented in [15].)

- Northeast U.S. blackout, November 1965: a threshold was exceeded that had been set too low, resulting in a 13-hour outage that affected New York, Pennsylvania, Vermont, Connecticut and Massachusetts.

- Lower New York State blackout, July 1977: recovery took in excess of 26 hours.

- Parts of ten Western U.S. states blacked out, October 1984: a faulty computer reading (the actual power loss was off by a factor of two) caused a chain reaction, although the triggering loss was reportedly an insignificant routine event that was supposed to have been handled seamlessly.

- Western U.S., July 1996: a heat-wave expanded power lines, which came in contact with a tree. Resulting power outages propagated over at least 12 states.

- Western U.S./Canada/Baja California (Mexico), August 1996: a summer heat-wave triggered cascading power outages, resulting in air-traffic slowdowns and many collateral problems.

- Northeast U.S., August 2003: a software race condition and an alarm-system failure caused a computer crash and widespread power outages; recovery took almost
2 days.

- Circuit breakers tripped in Maryland, in Queens, New York, and in Philadelphia on 25 May 2006, shutting down Amtrak, NJ Transit, and MARC trains in the morning rush-hour for the day. Five trains were stuck in tunnels.

- Queens, New York, July 2006: century-old wiring failed over a wide local region; the outage lasted a week.

What is perhaps most alarming about this list of outages is that such cases continue to occur. In some cases, this happens despite efforts to take remedial measures; in other cases, such measures are not even taken. Again, we often hear that a particular propagating outage is impossible because of the system design. Then, after it happens, we are typically told that the system or software or algorithms have been changed to prevent the problem from ever happening again. And then something similar happens again. Clearly, more proactive efforts are needed to analyze systems for potential widespread fault modes and ensuing risks, and to take appropriate timely actions. Of course, similar conclusions apply to proactive defenses against environmental disasters, as in the case of tidal waves, global warming, and anticipation of hurricanes (which was lacking in New Orleans before Katrina).

Leslie Lamport has remarked that a distributed system is a system in which you have to trust components whose existence is completely unknown to you. In the present context, distributed control of distributed systems with distributed sensors and distributed actuators is typically more vulnerable to propagated outages and other perverse failure modes such as deadlocks and other unrecognized hidden interdependencies (particularly on components that are untrustworthy and perhaps even hidden), race conditions and other timing quirks, coordinated denial-of-service attacks, and so on. Achieving reliability, fault tolerance, and system survivability in highly distributed systems is problematic; both formal analyses and system testing are highly desirable, but also potentially more complex than in nondistributed systems.

Furthermore, security, reliability, and system survivability are closely linked. A system is not likely to be secure if it is unreliable; similarly, a system is not likely to be reliable if it is not secure. A system is not likely to be survivable in the face of natural, accidental, and malicious adversities if it is not secure, reliable, fault tolerant, people tolerant, usable, and easily administered, with sufficient performance even under crises.

To illustrate this point, although the ARPANET outage and the AT&T long-lines collapse were triggered spontaneously by mechanisms that required no human intervention, each could have been equally well caused by (possibly remote) human activity, either intentionally (by anyone who knew about the fault mode) or accidentally. In the ARPANET case, the insertion of two bogus network status messages could have created the same effect as the dropped bits. In the AT&T case, inadvertent action by maintenance personnel or intruders on one telephone switch could have initiated the same sequence of propagating crashes. (There is some reason to suspect that this might actually have happened.) Similar reasoning also applies to the distribution of electrical power, where the control systems are heavily dependent on computer systems, with considerable functionality accessible from the Internet. Thus, security, reliability, and survivability need to be considered together rather than separately.

Any discussion of network disruptions must of course mention viruses, worms, propagating Trojan horses, and other forms of distributed malware. It is worth revisiting the 1988 Internet Worm [23, 29], which exploited four different system weaknesses — a buffer overflow in the `finger` daemon, systems configured with overly permissive `.rhosts` access, the `sendmail` debug option, and dictionary attacks on encrypted password files; however, it ran amok because of an overly aggressive algorithm for keeping the worm alive. Preventing malware might be aided by a system environment that provides a combination of sound computer system architectures with confined execution environments or virtual machines, rigorously controlled interactions among partitions (e.g., [3]), carefully configured fine-grained access control policies (e.g., [9]), good software engineering practice with safe uses of sensible programming languages and static analysis tools for software. With these and other approaches, the challenge of preventing exploitations of malware and security flaws such as buffer overflows could be much less problematic than it is today. (See [1, 10, 19, 31] for taxonomies of security flaws.)

With respect to confining the propagation of undesirable behavior in networks and distributed systems, the principles of security and good software development could contribute considerably. For example, the principle of least privilege suggests the development of architectures that enforce confined-domain or virtual-machine process isolation, with intercomponent interfaces and network protocols that are well defined, carefully controlled, and carefully analyzed. The principle of complete mediation suggests an architecture in which only a subset of the system needs to be highly trustworthy, and in which the integrity of that subset cannot be circumvented.

4. COMPUTERIZED ELECTIONS

The problems of trying to ensure the fairness, accuracy, and overall integrity of elections necessitate many requirements that span the entire election process. Various end-to-end requirements are needed to address system integrity, reliability, and survivability, vote integrity, overall accountability and oversight, nonsubvertible audit trails, impartial resolution of disputes, and uncompromised voter privacy (to identify just a few requirements). Simultaneously satisfying both vote integrity and voter privacy requires special care in design and implementation of operational procedures and relevant computer systems.

Since the mid-1980s, various efforts have been made to increase the automation of the voting process, particularly with the wider use of computer-based systems. (See Mercuri's doctoral thesis [11] for an extensive set of requirements casting the computer system requirements into the framework of the Common Criteria.) However, the need for trustworthy computing is only part of the problem. Trustworthiness is required throughout the end-to-end process.

Ideally, the total-process combination of computer system architectures and operational procedures should seek *strength in depth*, to prevent isolated failures or single-point attacks from compromising the end results. Unfortunately, the entire election process today represents *weakness in depth:* every step is a potential weak link — from registration to voter authentication to vote casting, vote counting, and transmission of the partial and final results (perhaps even via the Internet or wireless communications, perhaps unencrypted or weakly protected, and subject to denial-of-service attacks). Indeed, each stage in the voting process represents potential vulnerabilities that can be compromised in many ways — for example, accidentally or intentionally, detectably or undetectably, technologically or procedurally. The potential risks of faults, errors, failures, and misuse encompass human, environmental, and technological causes. Each step must be safeguarded from the outset with extensive cross-checks, and should be noncompromisibly audited. These requirements become especially important whenever election results are contested.

In today's all-electronic paperless voting machines, there is effectively no independent confirmation that the vote that is cast is the same as the vote that is counted, and that the results remain unchanged throughout. Furthermore, there is no independent audit trail that would enable an irrefutable recount. Today's unauditable proprietary touch-screen systems (e.g., [25]) are typically evaluated under a proprietary process that is commissioned by the vendors,

relative to voluntary standards that are inherently weak. The result is fundamentally unconvincing from the perspective of trustworthiness.

One potential remedy for having to trust today's unauditable proprietary all-electronic voting systems is to add an independent voter-verified paper trail that is the vote of record [11], as some vendors are now attempting to do. However, this introduces further complexities, and appears to be only a palliative addition to an already overly expensive PC-based solution when compared with less expensive paper-based alternatives that are much less dependent on untrustworthiness of computer systems, such as optical scan technologies (but which of course have their own potential risks).

Of great potential interest to security researchers are possible voting systems and associated procedures that are designed, implemented, and rigorously evaluated openly to be demonstrably able to satisfy stringent end-to-end requirements. One very promising direction involves cryptographic approaches (e.g., [2, 4, 12]) that could be formally verified. Of course, a fundamental challenge for the cryptography community is to provide convincing evidence that the resulting total systems (that is, not just the cryptography in isolation) are demonstrably resistant to internal tampering, external manipulation, and undetected system errors, including compromise from any underlying operating systems (which do not even have to be considered under today's voluntary evaluation guidelines) or from compiler subversions (such as in [30]). Such evidence would need to be convincing to independent cryptographers, system security experts, and — to a considerable extent — the general public.

Although the problem of adding up a bunch of votes might intuitively seem easy, the complexity of the overall election process is considerable, particularly in operational practice. With respect to computerized elections, serious observance of the principles of Section 2 could (in principle) provide dramatic improvements in today's voting technology, but would need to be applied end-to-end to detect, prevent, and recover from software-hardware failures and power outages, as well as surreptitious manipulations and other unexpected irregularities in human procedures throughout the election process.

Of particular relevance here is the principle of least privilege. As an extension of that principle, what is needed are system architectures, network protocols, and operational procedures that minimize the areas of vulnerability, for reasons similar to those in Section 3, but with rather different threat models. Also needed are independent cross-checks that can ensure the integrity of the process despite accidental failures and intentional manipulations. In addition, the principle of pervasive nonbypassable auditing is critical, with nonalterable audit trails and oversight sufficient to enable incontrovertible reconstruction of all votes as cast.

5. CONCLUSIONS

If we take the principled advice of Section 2 and the examples of Section 3 and Section 4 seriously, several benefits can result.

- The development process could be dramatically improved. Principled system developments and selective uses of formal methods are increasingly needed. Greater up-front emphasis on requirements, sound architectures, and good software engineering practice can result in fewer cost overruns, fewer delivery delays, easier system operation, and much less need for recurring patch management. Systems should be designed to be predictably composable out of evaluated subsystems, with correspondingly predictable properties.

- Many characteristic system vulnerabilities can be systematically avoided or at least considerably diminished, especially in newly developed systems, but also through judicious use of well-conceived static analysis tools applied to existing systems.

- Although there are always many difficulties that arise in trying to retrofit wisdom into legacy systems, many of the principles can also be applied to incremental evolutions — for example, incorporating an architecturally sound combination of old and new subsystems, or attempting rather humbly to encapsulate or otherwise mask the weaknesses of older systems (albeit with somewhat lowered expectations of trustworthiness).

Trustworthiness (especially with respect to overall security, reliability, system survivability, and human safety) is inherently a weak-link phenomenon. In poorly designed systems, one flaw may be enough to result in compromise. Simplistic solutions are likely to be untrustworthy. Merely patching a few of the more obvious flaws is not likely to be very successful. Therefore, considerable pessimism should accompany any efforts to make silk purses out of sows' ears. However, trying to minimize what must be trustworthy, avoiding violations of the principle of least privilege, and wisely applying some of the other principles can help significantly.

This paper addresses just the tip of a huge iceberg that is lurking in our path. It should not be surprising. Its message is clearly not novel (for example, see [14, 28]), but nevertheless still fundamental and still underappreciated. In addition, more of the less visible portions of the iceberg are examined in [16], along with how to attain predictable composability of trustworthy systems with some meaningful measures of assurance.

The increasing complexity of modern systems, the critical dependence on computer-communication technology, and the ever greater need for trustworthiness make it imperative that we reflect on mistakes of the past and attempt to avoid them in the future. The road to disciplined system development is clearly paved with good intentions, and the suggested course is riddled with practical pitfalls. However, designing and implementing systems with trustworthiness as an integral fundamental goal is a very important challenge, and is well worth pursuing vigorously in research and development practice.

6. ACKNOWLEDGMENTS

This paper was prepared in part under National Science Foundation Grant Number 0524111. The author thanks Douglas Maughan, who sponsored the work culminating in [16] when he was a Program Manager in the U.S. Defense Advanced Research Projects Agency (DARPA). (Doug is now a Program Manager in the Science and Technology Directorate in the Department of Homeland Security, where he spearheads research and development relating to cybersecurity.) The author is grateful to Rebecca Wright for her very helpful feedback.

7. REFERENCES

[1] R.P. Abbott et al. Security analysis and enhancements of computer operating systems. Technical report, National Bureau of Standards, 1974. Order No. S-413558-74.

[2] B. Adida and C.A. Neff. Ballot casting assurance. In *Workshop on Electronic Voting Technology Workshop*, Vancouver, BC, Canada, August 2006. USENIX.

[3] Steven M. Bellovin. Virtual machines, virtual security? *Communications of the ACM*, 49(10), October 2006. *Inside Risks* column.

[4] J. Benaloh. Simple verifiable elections. In *Workshop on Electronic Voting Technology Workshop*, Vancouver, BC, Canada, August 2006. USENIX.

[5] F.J. Corbató. On building systems that will fail (1990 Turing Award Lecture, with a following interview by Karen Frenkel). *Communications of the ACM*, 34(9):72–90, September 1991.

[6] F.J. Corbató, J. Saltzer, and C.T. Clingen. Multics: The first seven years. In *Proceedings of the Spring Joint Computer Conference*, volume 40, Montvale, New Jersey, 1972. AFIPS Press.

[7] E.W. Dijkstra. The structure of the THE multiprogramming system. *Communications of the ACM*, 11(5), May 1968.

[8] R.J. Feiertag and P.G. Neumann. The foundations of a Provably Secure Operating System (PSOS). In *Proceedings of the National Computer Conference*, pages 329–334. AFIPS Press, 1979. http://www.csl.sri.com/neumann/psos.pdf.

[9] P.A. Karger. Limiting the damage potential of discretionary Trojan horses. In *Proceedings of the 1987 Symposium on Security and Privacy*, pages 32–37, Oakland, California, April 1987. IEEE Computer Society.

[10] C.E. Landwehr, A.R. Bull, J.P. McDermott, and W.S. Choi. A taxonomy of computer program security flaws, with examples. Technical report, Center for Secure Information Technology, Information Technology Division, Naval Research Laboratory, Washington, D.C., November 1993.

[11] R. Mercuri. *Electronic Vote Tabulation Checks and Balances*. PhD thesis, Department of Computer Science, University of Pennsylvania, 2001. http://www.notablesoftware.com/evote.html.

[12] C.A. Neff. A verifiable secret shuffle and its application to e-voting. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 116–125, Philadelphia, Pennsylvania, November 2001.

[13] P.G. Neumann. Illustrative risks to the public in the use of computer systems and related technology, index to RISKS cases. Technical report, Computer Science Laboratory, SRI International, Menlo Park, California. Updated regularly at http://www.csl.sri.com/neumann/illustrative.html; also in .ps and .pdf form for printing in a denser format.

[14] P.G. Neumann. The role of motherhood in the pop art of system programming. In *Proceedings of the ACM Second Symposium on Operating Systems Principles, Princeton, New Jersey*, pages 13–18. ACM, October 1969. http://www.multicians.org/pgn-motherhood.html.

[15] P.G. Neumann. *Computer-Related Risks*. ACM Press, New York, and Addison-Wesley, Reading, Massachusetts, 1995.

[16] P.G. Neumann. Principled assuredly trustworthy composable architectures. Technical report, Computer Science Laboratory, SRI International, Menlo Park, California, December 2004. http://www.csl.sri.com/neumann/chats4.html, .pdf, and .ps.

[17] P.G. Neumann, R.S. Boyer, R.J. Feiertag, K.N. Levitt, and L. Robinson. A Provably Secure Operating System: The system, its applications, and proofs. Technical report, Computer Science Laboratory, SRI International, Menlo Park, California, May 1980. 2nd edition, Report CSL-116.

[18] P.G. Neumann and R.J. Feiertag. PSOS revisited. In *Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC 2003), Classic Papers section*, pages 208–216, Las Vegas, Nevada, December 2003. IEEE Computer Society. http://www.acsac.org/ and http://www.csl.sri.com/neumann/psos03.pdf.

[19] P.G. Neumann and D.B. Parker. A summary of computer misuse techniques. In *Proceedings of the Twelfth National Computer Security Conference*, pages 396–407, Baltimore, Maryland, 10–13 October 1989. NIST/NCSC.

[20] D.L. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12), December 1972.

[21] N.E. Proctor and P.G. Neumann. Architectural implications of covert channels. In *Proceedings of the Fifteenth National Computer Security Conference*, pages 28–43, Baltimore, Maryland, 13–16 October 1992. (http://www.csl.sri.com/neumann/ncs92.html).

[22] L. Robinson and K.N. Levitt. Proof techniques for hierarchically structured programs. *Communications of the ACM*, 20(4):271–283, April 1977.

[23] J.A. Rochlis and M.W. Eichin. With microscope and tweezers: The Worm from MIT's perspective. *Communications of the ACM*, 32(6):689–698, June 1989.

[24] E. Rosen. Vulnerabilities of network control protocols. *ACM SIGSOFT Software Engineering Notes*, 6(1):6–8, January 1981.

[25] A. Rubin. *Brave New Ballot*. Random House, 2006.

[26] J.M. Rushby. The design and verification of secure systems. In *Proceedings of the Eighth ACM Symposium on Operating System Principles*, pages 12–21, Asilomar, California, December 1981. (ACM Operating Systems Review, 15(5)).

[27] J.M. Rushby and B. Randell. A distributed secure system (extended abstract). In *Proceedings of the 1983 IEEE Symposium on Security and Privacy*, pages 127–135, Oakland, California, April 1983. IEEE Computer Society.

[28] J.H. Saltzer and M.D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, September 1975.

[29] E.H. Spafford. The Internet Worm: crisis and aftermath. *Communications of the ACM*, 32(6):678–687, June 1989.

[30] K.L. Thompson. Reflections on trusting trust. *Communications of the ACM*, 27(8):761–763, August 1984.

[31] K. Tsikpenyuk, B. Chess, and G. McGraw. Seven pernicious kingdoms: A taxonomy of software security errors. *IEEE Security and Privacy*, 3(6), November-December 2005.