Risks of Untrustworthiness

Peter G. Neumann
Principled Systems Group
Computer Science Laboratory
SRI International
Menlo Park CA 94025-3493
Neumann@CSL.sri.com

Abstract

This paper revisits the risks of untrustworthiness, and considers some incidents involving computer-based systems that have failed to live up to what had been expected of them. The risks relate to security, reliability, survivability, human safety, and other attributes, and span a variety of applications and critical infrastructures — such as electric power, telecommunications, transportation, finance, medical care, and elections. The range of causative factors and the diversity of the resulting risks are both enormous. Unfortunately, many of the problems seem to recur far too often. Various lessons therefrom and potential remedies are discussed.

1 Risks

This contribution to the Classic Papers track is a retrospective consideration of computer-related risks from the archives of the ACM SIGSOFT *Software Engineering Notes* (since 1976) and the online ACM Risks Forum (since 1985, a.k.a. RISKS and comp.risks), both of which were created by the author. The cumulative *Illustrative Risks* index to both sources [12] provides a hint of the enormous range of problems that must be considered. Discussion of many interesting cases prior to 1995 is found in [13]; surprisingly, apart from a steadily increasing number of more recent instances of similar cases, the basic conclusions of that book are still very timely!

Application areas in RISKS include space missions, defense, aviation and other forms of transportation, power, telecommunications, health care, process control, information services, law enforcement, antiterrorism, elections, and many others. The causes of computer-related risks are manifold. The RISKS archives include rampant cases of power glitches, undetected hardware failures, flawed software, human error, malicious misuse, questionable election results, and even animal-induced system failures. The results of these problems have caused deaths, physical injury and health problems, mental anguish, financial losses and errors, fraud, security and privacy violations, environmental damage, and so on. There is much to be learned from this litany of cases.

People are always a potential weak link, throughout the system life cycle. Although technology is sometimes blamed, people have created that technology. For example, requirements are often incorrect, incomplete, mutually inconsistent, and lacking in foresight. System designs and detailed architectures are typically flawed. Software is frequently buggy. Patches intended to fix existing flaws often create further bugs. System adminstrators are usually beset with too many opportunities for mistakes. Indeed, blame can often be spread rather widely.

2 Trustworthiness

The term *trustworthiness* implies that something is worthy of being trusted to satisfy its specified requirements. The requirements may specify in detail various system properties such as security, reliability, human safety, and survivability in the presence of a wide range of adversities. Trustworthiness thus implies some sort of assurance measures, and is typically *never* perfect.

Trustworthiness needs to be considered pervasively throughout the system life cycle, through system development, use, operation, maintenance, and evolutionary upgrades. It cannot be easily retrofitted into systems that were not carefully designed and developed. It is dependent on technology and on many other factors — the most important of which ultimately tends to be people.

Sections 3 through 6 discuss a few instructive cases of untrustworthiness, with references in [12, 13].

3 Unreliable Backup

A major source of problems relates to failures of backup systems, or failures of the interface between the primary and backup systems, or in some cases the total absence of backup.

One of the most interesting cases of a problem involving a backup system arose in NASA's very first attempt to launch a shuttle, the Columbia. The synchronization problem in the first shuttle was partly a design error and partly a programming flaw. About 20 minutes before the scheduled launch on 10 April 1981, the backup computer failed to be synchronized with the four primary computers. This failure had actually occurred previously in testing, and was later identified as a one-in-64 probabilistic intermittent [5], but was apparently not known to the operations crew. The two-day delay in launch could apparently have been avoided by a retry.

Several major airport disruptions are also worth noting, as well as other cases that resulted in total system failures, either because of the lack of a backup system or in spite of its presence.

Air-traffic control (ATC) backup/recovery failures:

- Palmdale (Los Angeles) ATC, July 2006: a pickup truck hit a utility pole; automatic cutover to backup power failed an hour later.
- Reagan National Airport, 10 April 2000: main power and backup failed for almost 8 hours; major outage.
- Westbury, Long Island ATC, June 1998: software upgrade failed its test, but reversion to the old software failed.
- Three main New York airports shut down, 1991: a Number 4 ESS telephone system had a 4-hour outage; the standby generator had been misconfigured, and the system ran (without the generator) until the backup batteries had been drained.
- Twenty ATC systems were shut down, 1991: a fiber cable was accidentally cut by a farmer burying his cow.
- El Toro (Los Angeles) ATC, 1989: 104 hardware failures occurred in a single day, with no backup system.

More total system failures and backup (or no backup!):

- Swedish central train-ticket sales and reservation system, 1998: hardware and backup system both failed for an entire day.
- Washington Metro Blue Line, 1997: main system and backup both failed, causing major delays.
- San Francisco Bay Area Rapid Transit, April 2006: software upgrade attempts failed for three days in a row, causing long delays. On the third day, backup was attempted to the previous system – which failed.
- Japanese stock exchange, November 2005: the primary system crashed, and the cutover to the backup system failed (it was using the same software).
- 9 Mile Point nuclear power plant in Oswego, NY, 1991: a power surge shut down the plant when the "noninterruptible" power supply failed.
- New York Public library, 1987: lost its computerized references, for which there were no backups.
- Dutch criminal management system, 1987: a new system failed, freed some criminals, caused arrest of others who were innocent; the old system had been eliminated, and no backup was possible.

Although these problems all relate to system survivability, security issues also arise in backup systems — including data integrity (particularly for forensic purposes and election system disputes), data retention, long-term compatibility of backup data, noncompromisibility of personal data, and privacy. Furthermore, if a system and its backup and recovery facilities are not reliable, they may also not be secure.

In addition, backup systems must be considered in the context of the overall systems in which they function. It is not very helpful to claim that a backup system works perfectly in isolation if it is never properly invoked and never tested in conditions of actual need. Various cases are noted of backup systems passing periodic tests and nevertheless failing in operation. (For example, a diesel generator stopped working because the fuel pump keeping its tank full depended on utility power — which had always been available during testing [2]!) Thus, backup systems must be demonstrably trustworthy with respect to their ability to satisfy criteria for security, integrity, reliability, and survivability (among other requirements), and must be tested under realistic conditions.

4 Unrobust Networks

Various examples of widespread propagation effects exhibit some of the complexities inherent in distributed and networked systems. Of particular interest to computer networks are two cases in which global failure modes resulted from local faults, namely the 1980 ARPANET collapse [20] and the 1990 AT&T long-distance collapse (e.g., see [13]). Also of interest are various massive U.S. power outages that resulted from an initial power blip propagating widely. Among major outages, the Northeast power blackout in November 1965 was followed by outages affecting 10 western states in October 1984, the Western U.S. in July 1996, Western U.S., Canada, and Baja Mexico in August 1996, and the Northeast in August 2003. These cases are revisited in [15], with references in [12]. Propagating malware (e.g., viruses and worms) such as the 1988 Internet Worm is also worth noting. Although malware may be a direct threat to systems connected to networks, it also may threaten the throughput and reliability of the networks themselves. In addition, natural causes may also cause widespread disruption, as in the case of Hurricane Katrina.

The above cases are illustrative of the unfortunate reality that the same kinds of failures continue to recur, despite efforts to avoid them. This is particularly true of the most frequent types of security flaws and propagating outages of computer networks and power grids. The need for real proactive measures is apparently subordinated by other demands.

One of the main lessons from these outages is that security, reliability, and survivability are closely interrelated, especially when there are people in the loop. For example, both the ARPANET outage and the AT&T long-lines extreme slowdown could have been easily triggered remotely by subversive human activity — if the fault mode had been known to the perpetrator — rather than accidentally.

5 Unsafe Systems

To security-minded people, many past incidents that resulted in accidental losses of life, injuries, and serious impairment of human well-being further illustrate the difficulties in providing high-assurance trustworthy systems. Many of the lessons that should be learned for human safety are rather similar to those in developing secure systems, and suggest that many commonalities exist between safe systems and secure systems.

- Requirements errors abound. Many critical systems are developed with no specified requirements, or perhaps incomplete ones.
- Design flaws abound. In many cases, the system architectures and detailed designs are inherently incapable of satisfying the intended requirements, although this is often not identified until much later in the development cycle.
- Programming bugs abound. (I once posed an exam question that could be satisfied with a five-line program. One student managed to make *three* programming errors in five lines, including an off-by-one loop count and a missing bounds check.)

A few pithy examples of safety-related risks are summarized here, particularly as a reminder to younger people who were not around at the time. References are found in [12].

- Aviation, defense, and space. The RISKS archives include many cases of deaths involving commercial and military aviation. Here are just a few examples. The Iran Air Airbus mistakenly shot down by USS Vincennes' Aegis missile system was attributed to human error and a poor human interface. The Patriot system defending against Iraqi scud missiles had a serious hardware/software clock drift problem that prevented the system from tracking targets after a few days. The Handley Page Victor tailplane broke off in its first high-speed flight, killing the crew. Each of three independent test methods had its own flaw that made the analysis appear satisfactory, which consequently prevented identification of a fundamental instability. A Lauda Air aircraft broke up over Thailand, after its thrust-reverser accidentally deployed in mid-air. A British Midland plane crashed after an engine caught fire and the pilot erroneously shut off the remaining good engine because the instrumentation had been crosswired. Three early A320 crashes were blamed variously on pilot error, safety controls being off, software problems in the autopilot, inaccurate altimeter readings, sudden power loss, barometric pressure reverting to the previous flight, and tampering with a flight recorder; in one case, the pilots were convicted of libeling the integrity of the technology! An Air New Zealand flight crashed into Mt. Erebus in Antarctica; computerized course data was known to be in error, but the pilots had not been informed. (Incidentally, the shuttle Discovery's tail speed-brake gears were installed backwards in 1984, but this was not discovered until 2004, 30 missions later!)

- Rail travel. The RISKS archives include dozens of train wrecks attributable to various hardware, software, and operational problems, some despite signaling systems and safety devices, some as a result of manual operation when automated systems failed.

- Ferry crashes. The Puget Sound ferry experienced numerous computer failures that resulted in twelve crashes, and the removal of the automated "sail-by-wire" system.

- Nuclear power. The Chernobyl accident in 1986 was the result of a misconceived experiment on emergency-shutdown recovery procedures. The long-term death toll among cleanup crew and neighbors continues to mount, 20 years later. The earlier Three Mile Island accident in 1979 was attributed to various equipment failures, operational misjudgment, and a software flaw (reported in 1982 by Daniel Ford [4]): experimentally installed thermocouple sensors were able to read abnormally high temperatures, but the software suppressed readings that were outside of normal range — printing out "???????" for temperatures above 700 degrees, and thus masking the reality of a near-meltdown in which temperatures reached over 4000 degrees.

- Medical care. The most often cited example of a software failure in a medical device is the Therac 25 accelerator, which resulted in several deaths as a result of a race condition in a nonatomic transaction of switching from the high-intensity research mode to the low-intensity therapeutic mode [10]. A physical interlock had been present in hardware in the Therac 20, and was mistakenly assumed to have been implemented in software in the Therac 25. At the other end of the technology spectrum was the heart-monitoring device with a standard electrical-socket wall plug instead of a jack, which had come loose; a cleaning person instinctively plugged it into the wall socket rather than into the monitor, thereby electrocuting the patient. Recent reports show new cases of operations on the wrong patient because of mistaken or misinterpreted computer data, erroneous test results, a mode-change fault in a glucose-monitoring device, and so on.

For each of these application areas (and many more), the safety risks of untrustworthy systems are considerable. In addition, risks tend to arise in supposedly safe systems with respect to security, privacy, reliability, system survivability, graceful degradation, and so on. Overall, much greater care is needed in developing safe systems than is devoted to run-of-the-mill software. (For example, see work by Leveson [7, 8, 9] for some serious approaches to enhancing safety that could be an inspiration to R&D in trustworthiness for secure applications.)

Safety-related accidents continue to occur, particularly in air, rail, and medical applications — e.g., caused by hardware/software malfunctions and errors by controllers, pilots, and operators. In hindsight, some of those should have been preventable with better human interfaces, cross-checking, adequate staffing, preventive diagnostics and maintenance, training, pervasive oversight, and so on.

6 Unsecure Systems

We next consider problems of unsecure systems, with the hopes of gaining some insights from the previous sections. For ACSAC, documenting historical and recent security vulnerabilities and their exploitations might seem to be preaching to the ACSACramental choir, and might cause me to be ACSACked or ACSACrificed for rampant repetitiousness. Of course, typical risks include penetrations by outsiders and misuse by insiders, e-mailstorms (e-maelstroms?) of spam and phishing attacks, and isolated and coordinated distributed denials of service, to name just a few. Vulnerable applications include a wide array of financial systems with potentials for fraud and undetected errors, databases with rampant opportunities for identity thefts and other privacy violations, all of the critical national infrastructures, electronic voting systems with serious needs for system integrity and voter privacy as well as detection and prevention of manipulations and errors, and many other types of systems.

Buffer overflows, bounds checks, type mismatches, and many other program flaws continue to appear, frequently causing security failures. There have been numerous efforts to provide a taxonomy for such problems (as for example [1, 6, 18, 22]), as well as efforts such as static analysis tools to detect the presence of the characteristic flaws. However, disciplined software development and systematic use of analysis tools are required. There are enormous needs for well-designed and well-implemented trustworthy systems that can satisfy a broad set of security requirements. Curiously, for many years, system integrity tended to be subordinated to confidentiality, whereas accountability remained more or less in the dark; preventing denials of service is often still widely ignored.

From a total-system perspective, it would be highly desirable that systems designed for security also be able to

satisfy some of the other requirements for trustworthiness that transcend security *per se*. For example, systems that are supposedly secure but unreliable may no longer be secure when unreliable. Similarly, supposedly secure systems that are not predictably survivable under certain environmental disruptions may become unsecure when transformed into fail-safe or other degraded operation. The concept of fail-secure systems presents some significant challenges.

7 Conclusions

Considering the broad scope of the problems considered here, including huge diversities among causes and effects, it should not be surprising that rather far-reaching measures are needed to prevent or even detect the emerging likelihood of risks relating to untrustworthiness. For starters, prevention and remediation of the risks must encompass better understanding of the full range of requirements, as well as better system architectures explicitly addressing those trustworthiness requirements with appropriate assurance, usability, operation, and maintainability. Greater attention needs to be devoted to the software engineering disciplines for implementing trustworthy applications, either based on underlying trustworthy infrastructures such as secure operating systems and networking, or alternatively able to architecturally surmount some untrustworthiness in subsystems. (Twenty-two paradigmatic approaches to building trustworthy systems out of less trustworthy components are examined in Section 3.5 of [14]. However, beware of compromises resulting from coordinated attacks or accidental misbehavior originating inside the implementations or in underlying layers of abstraction.) In addition, these concepts must be inculcated into the practice of developers and operational staff through enlightened education and training. Also valuable would be a stronger sense of corporate altruism and perhaps some intelligent government action. (However, with regard to legislation and regulation, be very careful what you ask for; you might get it, or — perhaps more likely — a badly distorted version of it).

Above all, trustworthiness demands a pervasive sense of systems in their entirety, considering the long-term risks in the global context of all relevant applications relative to the totality of all relevant requirements. The preceding four sections illustrate four types of systems in which greater trustworthiness is urgently needed: survivable backup systems, robust networking, safe systems, and secure total systems (such as networked operating systems together with all their networked applications). In each of these and many other application areas, the potentials for untrustworthiness must be considered with respect to the environments in which those systems operate. The desired trustworthiness properties are mostly emergent properties of the entire system, rather than isolated properties of subsystems. Nevertheless, a huge step forward in avoiding or circumventing untrustworthiness would result if the emergent properties of application systems as a whole could be systematically derived or otherwise inferred from the composable properties of the subsystems, and so on iteratively into lower layers of abstraction. For example, see the 1977 Robinson–Levitt paper [19] and its application to the Provably Secure System design [3, 16, 17], and Neumann's report on predictable composability [14].

Life-critical systems and supposedly secure systems should of course be held to higher standards than conventional software, although criteria and evaluations tend to be not very rigorous. As one rather sad example, today's electronic voting systems (e.g., [11, 21]) are held to much weaker standards than gambling machines!

Myopia is very dangerous with respect to trustworthiness. The commonalities among the different application areas considered here are likely to transcend would-be single-discipline solutions. Thus, a massive culture shift is needed to proactively develop and compositionally evaluate systems in their entirety and to assure their operational configurations and usability. This is of course especially important for applications with critical requirements for trustworthiness. The pleas for such approaches in many Classic Papers are old but nevertheless still timely.

Acknowledgments

The author thanks Douglas Maughan, who sponsored reference [14] when he was a Program Manager in the Defense Advanced Research Projects Agency (DARPA). Many of the concepts discussed there on how to develop composable high-assurance trustworthy systems and networks are relevant to avoiding risks such as those discussed here. This paper was prepared in part under National Science Foundation Grant Number 0524111.

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] K. Borg. Re: LA power outages. *ACM Risks Forum*, 24(39), 22 August 2006. http://catless.ncl.ac.uk/Risks/24.39.html#subj8.

[3] 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.

[4] D. Ford. *Three Mile Island: Thirty Minutes to Meltdown*. Viking Press, 1982. Sensor-related quote reproduced in *ACM SIGSOFT Software Engineering Notes, 11,* 3, 9–10, July 1986.

[5] J. Garman. The bug heard 'round the world. *ACM SIGSOFT Software Engineering Notes*, 6(5):3–10, October 1981.

[6] 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.

[7] N.G. Leveson. *Safeware: System Safety and Computers*. Addison-Wesley, Reading, Massachusetts, 1995.

[8] N.G. Leveson. A new accident model for engineering safer systems. *Safety Science (Elsevier)*, 42(4):237–270, April 2004.

[9] N.G. Leveson. A systems-theoretic approach to safety in software-intensive systems. *IEEE Trans. on Dependable and Secure Computing*, 1(1), January 2005.

[10] N.G. Leveson and C. Turner. An investigation of the Therac-25 accidents. *Computer*, pages 18–41, July 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] 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.

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

[14] 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.

[15] P.G. Neumann. System and network trustworthiness in perspective. In *Proceedings of the Thirteenth ACM Conference on Computer and Communications Security (CCS)*, Alexandria, Virginia, November 2006.

[16] 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.

[17] 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.

[18] 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.

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

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

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

[22] 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.