To Appear, IFIP World Congress '94
Link 3, Issue 5: How Much is Safety Worth?
Sess. L-29.2, 1 Sept 94

Hidden safety requirements in large-scale systems

Carl E. Landwehr

Naval Research Laboratory, Code 5542, Washington, DC 20375-5337, USA

Abstract

To avoid hidden safety problems in future large scale systems, we must be able to identify the crucial assumptions underlying the development of their components and to enunciate straightforward rules for safe component interconnection.

Keyword Codes: K.4.1; K.6.5; J.7
Keywords: Computers and Society, Public Policy Issues; Management of Computing and Information Systems, Security and Protection; Computers in Other Systems

1. THREE ACCIDENTS

Contact with the Mars Observer spacecraft was lost permanently August 21, 1993, after it was instructed to pressurize its propulsion system to enter orbit around Mars. Subsequent investigation[1] revealed that the most probable cause of the loss was the gradual leaking of oxidizer past a check valve during the spacecrafts eleven-month transit to Mars. Such a leak would have permitted fuel and oxidizer to mix in the piping of the unpressurized propulsion system, so that when the system was repressurized, the resulting reaction would have ruptured the pipes and caused the spacecraft to spin out of control. Millions of dollars invested in planned experiments were lost. The operational strategy adopted for this flight was based on similar strategies that had been used successfully, but only in near-earth, short term missions, where the resulting leakage would have been insignificant.

A SCUD missile struck a U.S. barracks in Dahran on February 25, 1991, killing 28 and injuring 98. A Patriot missile battery defending the area had failed to respond to the incoming missile. This failure was ultimately attributed to inaccuracy in converting its integer clock to a floating point representation. Only because the system had been run continuously for 100 hours was this this inaccuracy significant; original specifications of the Patriot system called for at most 14 hours of continuous operation[2].

The Therac-25, a computerized radiation therapy machine, became commercially available in 1982. Between 1985 and 1987, six accidents involving massive overdoses to patients occurred before the machine was recalled to make extensive design changes, including the installation of hardware safeguards against software errors[3]. Later study revealed that parts of the software design and some specific software routines used in the Therac-25 had been reused from the earlier Therac-20, which had incorporated hardware safeguards against overdoses. The removal of the hardware safeguards in the

Therac-25 combined with the reuse of the software permitted a non-safety-critical software flaw in the earlier system to become safety-critical in the later one.

While accidents involving large, complex systems almost invariably result from combinations of failures rather than single ones, there is a common thread in these three accidents: in each case, a system or procedure developed under certain assumptions, and which met those assumptions, was ultimately applied in a situation where those assumptions were knowingly or unknowingly violated, and this violation led to a catastrophic failure. Each of these accidents resulted in significant loss of property or life and is thus in the category of unsafe behavior. Each involves a large, complex system in which computers played a significant role.

2. WHAT IS SAFETY? SECURITY?

"Safety" is a word that most people understand intuitively. Providing a precise, technical definition for it is not so easy. Within the framework of dependability concepts developed by IFIP WG 10.4, a system may be considered safe if it avoids "catastrophic consequences on the environment,"[4] and in particular it avoids catastrophic failures. A system fails if it deviates from its specification; "catastrophic consequences on the environment" is evidently open to interpretation. Informally, a microwave oven controller that fails to activate the microwave source sufficiently to reheat tonight's leftovers would be a failure, but not catastrophic. One that fails to turn off the source when the timer elapses and consequently burns the food and damages the oven would have failed catastrophically.

In the U.S., the Department of Defense, Department of Energy, the Federal Aviation Administration, and the Food and Drug Administration, all define or embed various notions of safety in their directives and regulations. MIL-STD-882B, for example, defines safety as as freedom from conditions that can cause death, injury, occupational illness, or damage to or loss of equipment or property[5]. This gives a wide latitude for regulation, of course, perhaps unrealistically so, as Leveson observes[6]. She provides a clear explanation of safety in the established terms of system safety engineering: systems are modeled in terms of states and transitions; states that could, in combination with external conditions, lead to a mishap are identified as hazards. Safety-critical software functions are those that could directly or indirectly cause a hazardous state to exist.

Security, in the context of computers, has conventionally been defined in terms of protecting information against unauthorized disclosure, modification, or withholding (denial of service)[7], or, conversely, in terms of preserving its secrecy, integrity, and availability. These three terms (particularly "integrity") have themselves been the source of much discussion[8]. In the dependability framework, a system is secure if it prevents "unauthorized access to and/or handling of information"4. More recently, focusing on commercial applications, Parker[9] has argued that the purpose of information security is to preserve availability and utility, integrity and authenticity, and confidentiality and possession, and that any one of these properties can be lost independent of the others. Needham recently argued that in many cases, denial of service is in fact the security problem of primary concern[10].

3. WHERE DO SAFETY AND SECURITY OVERLAP?

Safety and security are closely related6. If we resort to the dictionary, we find that (at least in English), the roots of safety lie in the Latin salv(us): intact or whole. This root fits very well the notion of "safe" found in banks or secure military facilities; such a safe should remain whole in the face of attempts to break into it. "Secure" also derives from the Latin; its root is securus, meaning "apart from care" or care-free. So something both safe and secure should be intact and leave us unworried.

Few if any systems are built just to be either safe or secure. Invariably, the system is intended to perform some other function, and the safest or most secure system often would be one that never performed that function, be it driving a car or displaying a message, at all.

Safety, it has been argued, is an "emergent" property -- it emerges as a property of a system that cannot necessarily be identified in any specific single component. The same is true of security. Although one might think of a single, certified multilevel secure computer as secure in itself, if it is connected to another similar component, both may function entirely correctly, yet the composed system may be less secure than the individual systems were.

A distinction often suggested between the safety and security points of view is that security analyses must be concerned with intentionally (presumably maliciously) introduced faults (e.g., Trojan horses, viruses, worms), while safety analyses may assume a relatively benign environment and focus on the elimination of accidentally introduced faults. But this distinction weakens under examination. We do not normally expect that the air traffic controller will maliciously direct one plane at another, but we certainly want the air traffic control system to defend against behavior of that sort, intentional or not. And we may legitimately be as concerned about about flaws, malicious or otherwise, in the air traffic control programs as in systems protecting sensitive information.

A recently published set of documented security flaws includes many which were introduced accidentally but turned out to be exploitable by malicious users[11]. The threat that a user may invoke a Trojan horse has been a strong influence on computer security work -- but the user who actually invokes the Trojan horse (as opposed to its author) presumably does so accidentally, not maliciously. The notion of a Byzantine fault[12], which has received considerable attention in the safety community, offers an interesting parallel to that of the Trojan horse: in effect, it assumes that a device may fail (or keep operating) in a malicious way, providing different results to different requesters.

One particularly clear overlap between safety and security requirements occurs in the area of denial of service. If a system that is relied on to produce a critical piece of information -- perhaps control signals sent to a nuclear reactor or a railroad switch, but conceivably authentication information of some sort -- fails to produce it, a hazardous state may result.

4. UNDERSTANDING SECURITY FORMALLY

One of the methods used to develop secure computer systems has been to define security formally. The goal has been to obtain a definition of security that is simple and abstract enough that people can agree it is the property they want in the implemented system, and then to use that definition to guide, formally or informally, the system development.

Most formal models for security have focused on secrecy. The earliest models focused on access control, using state-machine models as a base7; later efforts have constrained information flow through restrictions on the traces of a system[13,14], and some recent work has applied information theory to model the capacity of covert channels[15,16]. Though there have been some attempts to apply secrecy models to treat fault-tolerance (hence denial of service)[17], this area is much less developed. Formal models have also been produced of protocols used to distribute cryptographic keys, in order to permit arguments to be framed about their security[18,19,20].

The "composability problem" in security is to identify a useful security property for individual components that would also hold for a system of such components properly connected[21]. Finding a composable security property that is also of practical interest has proven quite difficult, and the increasing prevalence of systems that are patched together from a variety of components has made this problem seem urgent[22,23]. NATO chartered a research study group to investigate the question, "How are the assurances associated with the trustworthiness of a composite system to be derived from the assurances associated with the subsystems?" and though it convened a workshop in the fall of 1991, results were inconclusive, and the group has been disbanded. Recently, McLean[24] has reported some progress.

5. HOW DO WE ASSURE THE SECURITY OF USEFUL APPLICATIONS?

Some of the earliest work in computer security[25] called for the construction of a "reference validation mechanism" that would bear fundamental responsibility for enforcing security (secrecy). This mechanism was to satisfy three requirements: it must be tamper proof, it must always (on every access made by a subject to an object) be invoked, and it must be "small enough to be subject to analysis and tests, the completeness of which can be assured." This formulation, which has provided the basis for much subsequent work in computer security, thus explicitly limited the size of the key security mechanism on the basis of what could be analyzed/tested completely.

The motive of this work was to support a large-scale, centralized, general-purpose, shared computing environment that would be able to separate users (potentially programmers) with different clearances and information with different classifications. Its approach is to isolate security-critical code and assure that it works as intended.

The Trusted Computer System Evaluation Criteria[26] (TCSEC) follow this approach: the Trusted Computing Base (TCB) is to incorporate all security-critical code. Applications should be able to be run outside the security perimeter and, because they do not enforce security requirements, they should not require security certification. Logically, they are layered on top of the TCB and subject to its security enforcement. For example, it should be possible to operate a database system on top of a TCB without difficulty. But in practice, this would limit the database to operating at a single security level at a time -- users who wished to put data from different security levels in the same database would be unable to do so without “upgrading” all of the lower level data to the highest level.

As it became clear that users would like their databases to provide a multilevel service, a need arose to provide evaluated database products as well as evaluated computer systems. This need eventually led to the Trusted Database Interpretation[27] (TDI). The writing of the TDI was difficult and contentious partly because two different approaches to providing trusted database management systems were being pursued. The first one[28] called for layering of database functions on top of an existing TCB. The previously evaluated TCB would be relied on to enforce its specified security policy, and the database system would provide additional layers that would refine that policy and apply it. The second approach[29] designated the database system as a “trusted subject” that might, for example, store relations in files that were at the highest security level of any tuple in the relation, but the database would be trusted to maintain labels within that file that would permit it to release less-classified parts of the relation to less-cleared users.

The TCB subsets approach is designed to permit “evaluation by parts”: each layer can be evaluated in succession; the underlying TCB does not require reevaluation when a new layer is added. However, this approach will likely require substantial reorganization of an existing commercial DBMS. The trusted subject approach, conversely, may not require much change to an existing database system, but because that system is effectively placed inside the security perimeter of an existing TCB, it is really necessary to evaluate the combination of the two systems, database and operating system, together. Evaluation by parts is not possible.

The original notion that we could have useful application system security by developing a strong, simple mechanism at the center of the system and simply layering the application on top of it seems to require some revision.

6. A DIFFERENT WAY TO FACTOR THE PROBLEM

Many mathematical results are available for the analysis of individual queuing systems: different arrival and service time distributions, different priority and service disciplines, and different queue capacities all have been studied. But when individual queues are combined into a network of linked queues, the analysis is greatly complicated. A key result, achieved about 20 years ago[30], showed how, if certain constraints were imposed on the queues in a network, the results of separate analyses of the individual queues could be combined simply to yield the solution for the network.

Similarly, we may carefully develop an individual computing system so that we have the needed assurance that it meets its specified safety/security requirements. But when we link separately developed systems, unanticipated new security or safety problems may occur. We need to identify principles or constraints like those identified by queuing theorists that will permit us to connect systems and understand their behavior.

One such principle for composing information systems has been developed at NRL in the Secure Information Through Replicated Architecture (SINTRA) project. Although it was developed strictly to meet military security needs, and its extension or adaptation to meet safety requirements has not been considered, we offer it here in the hope that it may provoke others to find similar principles.

In the SINTRA architecture[31,32], physical separation and data replication are used to provide an MLS database service. There are two kinds of components: single level systems, which are not relied upon to separate data at different classification levels, and replica controllers (RCs), which coordinate updates and must meet specific security requirements. The idea behind the architecture is that data entered at lower security levels (on single-level systems) are automatically replicated on higher security level systems. A user operating at a given security level deals with a system that contains all data that user is authorized to see. Any changes a user makes to data, the RC automatically propagates to higher level databases. Algorithms developed for the RC insure that the databases at different security levels are kept consistent and that covert information flow among them is constrained.

SINTRA's composition principle is simple: if a higher level system requires access to data originally classified at a lower level, then it must only access replicas of those data at its own level. In a world of SINTRA replica controllers and system-high systems, each operating at a particular security level, two systems can be connected by a replica controller without compromising confidentiality, because the replica controller only permits the upward flow of data. The SINTRA world is perhaps more restricted than the worlds treated by computer security theorists, but those restrictions make the approach practical.

7. DISCUSSION: THE NEED TO KEEP ASSUMPTIONS SIMPLE AND IN VIEW

We seem to have come a long way from our opening examples. How do the safety problems they reveal relate to architectures for MLS database service? The Mars Observer example does not even involve computers very directly! But all three of the examples do involve the application of systems -- procedural or computing -- in domains that were outside the set for which they were originally designed.

Similarly, the application of commercial database systems to enforce rigorous military security policies stretches those systems beyond their original limits. Sometimes, we may be able to reengineer existing systems so that their domain of application is extended directly, but we also need approaches like SINTRA that take account of the known limitations of systems but provide an environment in which they can be safely used to meet new needs.

Both for safety and security purposes, solid boundary walls are sometimes needed. One of the lessons of two decades of computer security research and development is that it is very difficult to build such walls within a single computer system and have justifiable confidence that they lack holes. Both the difficulty of structuring systems to permit evaluation by parts and the requirement that the entire combination of operating system and application be reevaluated under the trusted subject approach reflect this fact.

The declining cost of hardware and the straightforward assurance provided by properly organized physical separation of components are making architectures based on physical separation more attractive. The separation kernel approach suggested by Rushby and Randell in the early 1980s took this approach[33]; SINTRA has effectively developed those ideas to provide an effective, high assurance MLS database service without requiring either substantial reconfiguration or reevaluation of existing software.

What future large scale, complex applications are on the horizon that may have hidden safety requirements, and how might their safety requirements be exposed?

A fungus covering several acres underground in upper Michigan has been touted as the worlds largest organism. In this case, surely the Internet qualifies as the worlds largest computing system. What hidden safety requirements may it have?

The Internet worm[34] provided a dramatic example of a denial of service attack. But society is not (yet) dependent enough on the Internet for a denial of service attack on it to raise the same safety concerns that an attack on, for example, an emergency telephone response system (e.g. a 911 system in the U.S.) would do. Increasingly attractive services, however, which depend on users unknowingly retrieving remote programs and executing them on their local machines, will be hard to resist but may bring with them substantial security risks (e.g. the recent security flaw in Unix implementations of the Mosaic Internet browser[35]).

If we want to avoid hidden safety problems in future large scale computing systems, we must be able to identify straightforward rules to control the safe interconnection of components, and we must be sure that each component is operated within the scope of the assumptions that controlled its development. Until we are able to do this, we must expect hidden safety requirements to continue to manifest themselves in accidents.

REFERENCES

1. Report of the Mars Observer Mission Failure Investigation Board to the Administrator, NASA, Volume I, Dec., 1993, Washington DC.
2. *ACM Software Engineering Notes, 16* (3) p. 19 (July 1991).
3. Leveson, N. and C.S. Turner. An investigation of the Therac-25 accidents. CSE Dept. TR #92-11-05,Univ. of Washington, Seattle, WA, Nov. 1992.
4. Laprie, J.C. (ed.) *Dependability: Basic Concepts and Terminology*. ISBN 0-387-82296-8, Springer-Verlag, New York-Wien 1992.
5. US Dept. of Defense. MIL-STD-882B 1984. System Safety Program Requirements.
6. Leveson, N. Software safety: what, why, and how. *ACM Comp. Surveys* 18,2:125-164.
7. Landwehr, C. Formal models for computer security. *ACM Comp. Surveys 13*, 3:246-278.
8. Sandhu, R. On four definitions of data integrity. In *Database Security VII, Status and Prospects*, Keefe and Landwehr, eds., IFIP Trans A-47, North-Holland, 1994:257-268.
9. Parker, D. Crypto and the avoidance of information anarchy. Panel session, 1st ACM Conf. on Computer and Communication Security, Nov., 1993.
10. Needham, R. Denial of service. In *Proc. 1st ACM Conf. on Computer and Communication Sec.*, ACM Press, NY, ISBN 0-89791-629-8, Nov., 1993. pp.151-153.
11. Landwehr, C., A.R. Bull, J.P. McDermott, and W.S. Choi. A taxonomy of computer program security flaws, with examples. NRL Report NRL/FR/5542--93-9591. Naval Research Laboratory, Washington DC, 1993 (to appear, *ACM Computing Surveys*).
12. Lamport, L., R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Trans. on Prog. Lang. and Sys.4* (3):382-401 (July 1982).

13. Goguen, J. and J. Meseguer. Security policies and security models. Proc. 1982 IEEE Symp. on Security and Privacy, IEEE CS Press, 1982. Reprinted in *Proceedings of 1980-84 Symp. on Sec. and Priv., Vol. I*, IEEE CS Press, 1990, ISBN 0-8186-8999.
14. McLean, J. Security models and information flow. *Proc. 1990 IEEE Symp. on Research in Security and Privacy*, IEEE CS Press, 1990. pp.180-189.
15. Gray. J. On analyzing the bus-contention channel under fuzzy time. *Proc. Computer Security Foundations Workshop VI*, IEEE CS Press, ISBN 0-8186-3950-4, 1993.
16. Kang, M. and I. Moskowitz. A pump for rapid, reliable, secure communication. *Proc. 1st ACM Conf on Comp and Comm. Sec.*, ACM Press,NY,ISBN 0-89791-629-8:119-129.
17. Odyssey Research Associates Inc. Romulus Theories, Vol. III. (Mar, 1994).Ithaca, NY.
18. Burrows, M., M. Abadi, and R. Needham. A logic of authentication. *ACM Trans. on Comp. Sys.8* (1):18-36 (Feb. 1990).
19. Syverson, P. Knowledge, belief, and semantics in the analysis of cryptographic protocols. *J. Computer Security 1* (3-4):317-334, IOS Press, Amsterdam (Sept. 1992).
20. Kemmerer, R.A., C. Meadows, and J.K. Millen. Three systems for cryptographic protocol analysis. *J. Cryptology 7* (2) (1994).
21. McCullough, D. A hookup theorem for multilevel security. IEEE TSE 16 (6):563-568.
22. Lubbes, H.O. COMPUSEC: a personal view. *Proc. 9th Ann. Computer Security. Applications Conf*. Orlando, FL, IEEE CS Press, ISBN 0-8186-4330-7, Dec., 1993.
23. Landwehr, C. How far can you trust a computer? *SAFECOMP '93*., J. Gorski, ed., Springer Verlag, NY, ISBN 0-387-19838-5, 1993, pp. 313-326.
24. McLean, J. A general theory of composition for trace sets closed under selective interleaving functions. Proc. 1994 IEEE Symp. on Research in Security and Privacy, IEEE CS Press, 1994,pp. 79-95.
25. Anderson, J.P. Computer security technology planning study, Vol I. ESD-TR-73-51, AF Sys. Com., Hanscom AFB, Oct., 1972, avail as NTIS AD-758206.
26. DoD 5200.28-STD. Trusted Computer System Evaluation Criteria. Dec., 1985.
27. NCSC-TG-021. Trusted Database Management System Interpretation. April, 1991. National Computer Security Center, Ft. Meade, MD.
28. Lunt, T.F. Multilevel database systems: meeting class A1. In *Database Security, II: Status and Prospects*, North Holland, ISBN 0 444 87483 6 (1989) pp.177-186.
29. Wilson, J. A security policy for an A1 DBMS (a trusted subject). *Proc. 1989 IEEE Symp. on Research in Security and Privacy,* IEEE CS Press, ISBN 0-8186-1939-2, 1989, pp.116-125.
30. Baskett, F., K.M. Chandy, R.R. Muntz, and F.G. Palacios. Open, closed, and mixed networks of queues with different classes of customers. *J ACM 22* (2):248-260 (1975).
31. Froscher, J.N., and C. Meadows. Achieving a trusted database management system using parallelism. In *Database Security II: Status and Prospects*, North Holland, 1989.
32. Kang, M., O. Costich, J.N. Froscher. A practical transaction model and untrusted transaction manager for a multilevel secure database system. In *Database Security, V:, Status and Prospects*,(Thuraisingham and Landwehr, eds. IFIP Trans. A, North-Holland, 1993.
33. Rushby, J. and B. Randell. A distributed secure system. *IEEE Computer 16* (7):55-67.
34. Spafford, E.H. Crisis and aftermath. *Comm. ACM 32*(6):678-687 (June 89).
35. Bina, E. Telnet URL security problem: details. Available via NCSA Mosaic URL http://south.ncsa.uiuc.edu/telnet-details.html (as of 20 April 1994)