Achieving Principled Assuredly Trustworthy Composable Systems and Networks

Peter G. Neumann
Computer Science Lab, SRI International
Menlo Park CA 94025-3493
Neumann@CSL.SRI.com

Abstract

Huge challenges exist with systems and networks that must dependably satisfy stringent requirements for security, reliability, and other attributes of trustworthiness. Drawing on what we have learned over the past decades, our CHATS project seeks to establish a coherent common-sense approach toward trustworthy systems. The approach encompasses comprehensive sets of requirements, inherently sound architectures that can be predictably composed out of well-conceived subsystems, highly principled development techniques, good software engineering disciplines, sound operational practices, and judiciously applied assurance measures. Although such an approach is likely to seem completely old-hat to some researchers and totally impractical to commercial developers, the wisdom thus embodied is seldom used consistently (if at all) in practice; if it were used wisely, much of the untrustworthiness in today's systems would simply disappear. This paper briefly summarizes our approach and its potential benefits.

1. Introduction

We summarize the primary task of our two-year project under Contract N66001-01-C-8040, as part of DARPA's Composable High-Assurance Trustworthy Systems (CHATS) program. The DARPA Program Manager is Dr. Douglas Maughan. Our project spans the goals of the DARPA CHATS program — including trustworthiness, composability, and assurance. The final report [10] is available on the Web, in draft form prior to June 28, 2003, and in final form thereafter:

http://www.CSL.SRI.com/neumann/chats4.html
http://www.CSL.SRI.com/neumann/chats4.pdf
http://www.CSL.SRI.com/neumann/chats4.ps

In our project, we are confronting an extremely difficult problem — namely, how to attain demonstrably trustworthy systems and networks that need to operate under stringent requirements for security, reliability, survivability, and other critical attributes. In particular, we pursue sound foundations for the creation of trustworthy systems and networks that can be easily composed out of subsystems and components, with predictably high confidence that they meet their requirements and also do something sensible when forced to operate outside of the expected normal range of operational conditions. Toward this end, we examine various *principles* for achieving trustworthiness, consider *constraints* that might enhance composability, propose *architectures* and *trustworthy subsystems* that are inherently likely to result in trustworthy systems and networks, consider constraints on *administrative practices* that can reduce the risks of bad behavior, and explore approaches that can significantly increase *assurance.* We also outline needs for *new research and development* that could significantly improve the future for dependably trustworthy systems.

With respect to the future of trustworthy systems and networks, perhaps the most important recommendations involve the urgent establishment and consistent use of realistic highly disciplined and principle-driven architectures, as well as development practices that systematically encompass trustworthiness and assurance as integral parts of what must become coherent development processes and sound subsequent operational practices. Only then can we have any realistic hopes that our computer-communication infrastructures — and consequently our national infrastructures — will be able to behave as needed, in times of crisis as well as in normal operation.

Developing systems with critical trustworthiness requirements is inherently more complicated than ordinary software. The risks are typically much greater [9], and the challenges have no simple turn-the-crank solutions. Ability, understanding, experience, education, and enlightened management are crucial. Success can be greatly increased in many ways, including dependable hardware, robust system and network architectures, pervasive use of good software engineering practices, careful attention to human-oriented interface design and especially ease of operation and system administration, sound and properly used programming languages, trustworthiness-enhancing compilers, techniques for increasing interoperability among heterogeneous systems and subsystems, methods and tools for analysis and assurance, and other factors. The absence or relative deficiency of each of these factors today represents a potential weak link in a process that is currently riddled with too many weak links. On the other hand, much greater

emphasis on these factors can result in substantially greater trustworthiness, often with more predictable results.

The approach of our project is strongly inspired by historical perspectives of fruitful research efforts and extensive experience (both positive and negative) relating to the development of trustworthy systems. It is motivated by the practical needs and limitations of commercial developments as well as by CHATS successes in inserting greater discipline into the open-source world. It provides useful guidelines for disciplined system developments. It also identifies various recommendations for future efforts. As a consequence of the inherent complexity associated with the challenges of developing and operating trustworthy systems and networks, it is impossible to represent the breadth and depth of scope of our work in this brief summary, which merely touches on the problems and the potential solutions. Thus, we urge you to read the cited report [10] and see how much of it might be applicable to you.

The main thrust of the effort considers lessons drawn from past research and prototype developments, and various approaches that can lead to much greater trustworthiness. Many of these concepts will be well known to you, at least *in the small*; however, they need to be applied vigorously and consistently *in the large.* The would-be lessons of the past have been widely ignored, especially in commercial developments — with a multitude of excuses being offered: for example, perceived irrelevance of trustworthiness in the mass marketplace; lack of customer demands for systems satisfying critical requirements; absence of legal liability for consequential damages; difficulties in using good software engineering practices, high-assurance techniques, and evaluation criteria; legacy compatibility constraints; weak system-oriented computer science education; and a pervasive tendency to blame users and system administrators for the human failures that produce inherently flawed systems, weak network protocols, and badly designed human interfaces. Short-sightedness abounds. In particular, the perceived importance of short-term cost increases and delayed deliveries could be largely marginalized if the long-term costs of debugging, integration, recoding, incremental maintenance, frequent upgrades, and critical dependence on highly skilled administration were included in the analysis. It is widely recognized that up-front efforts can greatly reduce the overall effort.

We do not wish to preach to the choir or to proselytize the unwashed. This summary paper is written in the hopes that you will read the report, irrespective of your software development predilections. For developers who seriously seek to overcome the typical practice of seat-of-the-pants perhaps-just-good-enough software creation, particularly for critical applications, we offer a few caveats about the report. If you are looking for overall simplicity, you may be sadly disappointed; indeed, each chapter of the report may convince you that there are few if any easy answers. However, there are many answers — if properly applied. If you are looking for some practical advice on how to develop systems and networks that are substantially more trustworthy than what can be achieved today by patching together off-the-shelf flawed systems, then you may find some encouraging directions to pursue. We believe that diligent observance of the approaches described in our report [10] can greatly improve the situation. The opportunities for this within the open-source community are considerable; they are also potentially applicable to closed-source proprietary systems — if the excuses noted above can be eschewed. However, in any case, what is needed is greater discipline and attention to the fundamentals we discuss relating to the entire life cycle, including the presence of meaningfully comprehensive critical requirements, sound system and network architectures, highly principled software development, and operationally aware development. This approach is neither simplistic nor overwhelmingly complicated.

2. Approach of the Project

Our starting point involves the desired roles of principles relevant to the conceptualization, design, implementation, operation, and use of information systems and networks having critical requirements for security, reliability, and survivability. We also identify obstacles to achieving facile composability and interoperability, and to consider approaches that can contribute to the development of significantly greater composability in systems with critical requirements. We then consider characteristics of architectures that are likely to predictably satisfy the CHATS goals, based on the discussion of principles and the analysis of composability. If serious measures of assurance of the required trustworthiness are desired, then a variety of coherent and mutually supporting techniques needs to be applied. There is no one approach that fits all. Whenever assurance techniques are used, their strengths and limitations need to be understood from the outset, and applied specifically where they can be most effective. Various techniques are considered.

3. Attaining Trustworthiness

We believe that many R&D directions are important for the short- and long-term future — for computer and network communities, for developers, and for DARPA. The basis of our project is the exploration of a few of the potentially most timely and significant approaches, which are summarized roughly as follows. (See [10] for extensive discussion of each item.)

Principles. We revisit fundamental principles of trustworthy system development, cull out those likely to be most effective, explore their practical limitations, and provide a basis for principled architectures, principled development, and principled operation. For example, the 1975 Saltzer-Schroeder principles [16] are still mostly relevant today.

On one hand, some of the most basic principles seem to be treated merely as arcana from the ancient literature; they are found occasionally in university curricula, and are intuitively appealing to some developers. On the other hand, these principles are surprisingly absent in commercial programming practice, and therefore unobserved in system architectures, system implementations, programming languages, compilers, software engineering disciplines, and software development tools.

Certain principles are of great potential benefit to composability, trustworthiness, and assurance; these include (among others) abstraction, modularity, encapsulation, layered protection, separation of policy and mechanism, and separation of concerns — to the extent that they are sensibly applied to the architecture and the implementation, and that they do not interfere with one another. In practice, some of the principles can sometimes be contradictory, and therefore should not be overendowed or considered as ends in themselves.

Composability. We explore obstacles to achieving seamless composability and techniques for attaining practical composability in the future. Composability is meaningful at many layers of abstraction, for components, subsystems, networked systems, and networks of networks. It is also applicable to policies, protocols, specifications, formal representations, and proofs. Subsystem composability takes on a variety of forms, including sequential (e.g., with or without feedback, and with or without recursion) and parallel execution.

Composability has long been a crusade of the research community, but has typically been hindered by shortsighted development practices. Obstacles to seamless composability typically include (for example) hidden state interactions; emergent properties that exist beyond what is relevant to constituent subsystems and that manifest themselves only because of composition; nonmodular and poorly designed subsystems that seriously hinder decomposition; and compositions that simply do not scale properly. These and further difficulties often arise because of poor design abstraction and the lack of appropriate modularity and encapsulation. Thus, principled architecture and software development both can play a significant role in enhancing composability. Particularly relevant are the techniques and development tools that induce software engineering discipline, sound programming-language constructs, execution compatibility, and interoperability. However, in certain cases, compositions of seemingly compliant subsystems can actually compromise the ability of the resulting system to satisfy its requirements.

Trustworthy composable architectures. We characterize composable open distributed-system network-oriented architectures capable of fulfilling critical security, reliability, survivability, and performance requirements, while being readily adaptable to widely differing applications, different hardware and software providers, and changing technologies. By architecture, we specifically mean both the structure of systems and networks and the design of their functional interfaces (at each layer of abstraction).

Although different architectures may be needed for different classes of applications, they can share many common principles and attributes. We consider several alternative architecture approaches. Particularly appealing are architectures in which great attention is paid to minimizing the extent to which all subsystems must be trusted, in which trustworthiness can be concentrated primarily in a few particularly critical components within layered and distributed systems, with emphasis (for example) on trustworthy servers and highly constrained well-defined interfaces that can greatly enhance composability and interoperability. Ideally, we favor architectures in which many components do not have to be completely trustworthy (and in which some may be completely untrustworthy, as in Byzantine agreement), but where the overall system can still be adequately trustworthy. An example of such an architecture is a distributed and networked system in which a few really critical special-purpose components (e.g., dedicated servers) serve particular purposes for which they are extremely trustworthy. It should be possible to compose those special-purpose systems out of just a few trustworthy components, and to omit the vast majority of the bloatware that might otherwise be found in a conventional operating system together with its default applications. The clue to the *decomposition* problem (getting rid of all the functionality you do not need, and consequently getting rid of most of the corresponding security vulnerabilities) lies in the *composition* problem — that is, being able to constructively include just the essential functionality and nothing else (achieving a *stark subset*). Examples of such architectures are provided by Rushby and Randell [15] and Proctor and Neumann [13] in implementing multilevel security without having to trust every end-user system to enforce multilevel security (MLS), although similar approaches are valid for single-level systems. A different example is found in SeaView [7], where a high-assurance MLS database management system was demonstrably achievable by putting Oracle on top of an MLS security kernel, with no MLS-

trustworthiness required in the DBMS. Such architectures are needed to provide alternatives to the relatively undisciplined mass-market operating systems in which essentially all of the kernel code, utility programs, application code, test software, and user behavior must be trusted — even when they are not trust*worthy.* Also of considerable interest is the concept of combining subsystems in ways that actually increase the resulting trustworthiness; we enumerate 22 such basic trustworthiness-enhancing mechanisms in [10].

For truly complex sets of requirements, complexity can still be managed with a combination of sound composable architectures, hierarchically layered and horizontally distributed abstraction, sensible modularity together with encapsulation, stark subsetting for critical subsystems by removal of unnecessary functionality, proactively evolvable but well-conceived conceptually simple interfaces that mask local complexity, and principled software engineering. As a result, inherently complex systems can be reduced to relatively simple interconnections of relatively simple components with relatively simple interfaces and relatively simple exception conditions. (For a historical example of this, see the Provably Secure Operating System design [4, 11], in which each module was formally defined in a few pages of formal specifications, and the interlayer dependencies were also defined in a few lines of formally defined abstract implementations [11, 14], but where the overall system was quite complex.)

Trustworthy foundations and assurance. We seek sound bases for requirements, specification, implementation, trustworthiness, and assurance for predictably composable interoperable components. Whatever assurance measures are deemed desirable in increasing trustworthiness (as opposed to merely being trusted), those measures need to be distributed throughout the development process — from conceptualization and requirements through design, implementation, and operation — and integrated thoroughly throughout.

Some aspects of formal methods are certainly relevant, especially when critical requirements are involved. Perhaps surprising to some people, significant progress has been made in the past 30 years in formal methodologies and supporting tools, particularly for applications requiring safety, reliability, and security. One area of particular relevance that was explored in the 1970s but that has never been widely used is the ability to formally map hierarchical abstractions at one layer onto abstractions at other layers (as for example in [12, 14]). This approach could be extremely valuable in the present context, as it can enable the formal analysis of compositions (e.g., systems of systems).

The use of formally or semiformally based analytic tools is a very promising area. As one recent example, under our CHATS project, Hao Chen and Dave Wagner at the University of California at Berkeley have developed a model-checking environment that examines source code for presence or absence of certain types of characteristic implementation flaws in C code (e.g., involving setuid-like calls and root privileges); they have already applied this tool to sendmail, OpenSSH, and wu-ftpd [1, 2], and are continuing to extend its applicability.

Looking to a future in which special-purpose and general-purpose applications might become routinely composable out of more-or-less compatible demonstrably trustworthy components, an analysis tool would be highly desirable that can analytically determine the composability (among other properties) of software components — not just for the initial creation of a composed system, but also in subsequent reconfigurations, upgrades, and even dynamic installation of mobile code. This approach could (for example) take advantage of specifications and software formally shown to be consistent with those specifications, including descriptors relating to previously evaluated characteristics of the components (such as internal locks that might cause deadly embraces in certain contexts, assumptions regarding dependence orderings, identified interface limitations, and other attributes that might affect the compositionality). Ideally, this approach could then be used iteratively — for example, initially pairwise or n-wise, and then over successively wider scopes, possibly ascertaining obstacles to the desired compositions, or even potential failure modes that would suggest that a specific composition should not be permitted because of its identified deficiencies. Other properties could also be included, such as dynamic trustworthiness, configuration stability, and operational factors. We realize that there are all sorts of limitations of such an approach, but even small steps forward could be very useful.

Trustworthy protocols. We need to develop new protocols and/or extend existing protocols that effectively mask the peculiarities of various networking technologies wherever possible, but able to accommodate a wide range of technologies (e.g., wireless and wired, optical and electronic), and capable of addressing all relevant critical requirements. This is a very difficult challenge, and necessitates the involvement of NIST standards efforts, the development communities, and organizations such as the Internet Engineering Task Force (IETF).

Principled operational practice. We need to bring the above concepts into the realm of operational practice, which is seriously in need of greater dependability and controllability, including perhaps some formal dynamic analyses inline with would-be reconfigurations and dy-

namic system changes. Many of the concepts considered here have considerable potential toward that end.

4. Historical Reflections

Throughout the history of efforts to develop trustworthy systems and networks, there is an unfortunate shortage of observable long-term progress. Significant research and development results are typically soon forgotten or else widely ignored in practice. Computer systems have come and gone; programming languages have come and (occasionally) gone; certain specific systemic vulnerabilities have come and gone. However, many generic classes of vulnerabilities seem to persist forever — such as buffer overflows, stack mismanagement, race conditions, off-by-one errors, mismatched types, divide-by-zero crashes, and unchecked procedure-call arguments, to name just a few classes within a very long list. Overall, it is primarily only the principles that have remained inviolable — at least in principle — despite their having been widely ignored in practice. It is time to change that unfortunate situation, and honor the principles. This is particularly frustrating to researchers who know better, but should also be very embarrassing to commercial developers who don't (although it seems not to bother them very much).

As an example of lessons that might have been learned from the past, consider the Multics development that began in 1965, with considerable support from ARPA, MIT, Bell Labs, and Honeywell. Multics made some extremely important early advances. Its architecture included truly independent segmentation, paging, and concentric protection rings in the hardware, hierarchically structured directories, access-control lists, and dynamic linking of symbolic file names, and a highly principled development process that enforced approval of specifications before implementation and the use of a higher-level programming language (a subset of PL/I). See also Fernando Corbató's Turing lecture [3] for a survey of lessons learned from the effort. The system architecture took significant advantage of abstraction, modularity, and encapsulation, and of the constraints imposed by PL/I's handling of strings, arrays, and structures. Because of the flexibility built into the design, it was subsequently relatively straightforward to retrofit multilevel security into the delivered production version of the system. Karger and Schell have recently reconsidered their 1974 Multics security evaluation [5] with a fresh analysis of the evaluation experience. Their 2002 paper [6] notes (for example) that the use of PL/I, the underlying hardware protection, and the stack discipline almost completely avoided buffer and stack overflows, data interpretable as executables (because of the absence of the execute bit), Trojan horses, and other characteristic security problems — and generally greatly enhanced the security and the ease of evaluation. Unfortunately, many of the lessons observed by Corbató and more recently by Karger and Schell — and many of the principles enumerated by Neumann [8] in 1969, Saltzer and Schroeder [16] in 1975, and others — have been widely ignored in more recent system developments. DARPA should take considerable credit for ARPA's vision during the 1960s in supporting the Multics effort, but should now also recognize the importance of the lessons that should have been learned by others along the way to the present — and should encourage greater observance of those lessons. (Many additional references are given in [10].)

5. Directions for the Future

Much effort remains in demonstrating the practical relevance of the outlined approach. If intelligently applied, significant long-term benefits are likely to result, including inherently sound system and network architectures that minimize the dependence on the trustworthiness of all systems, subsystems, and users (local or remote, whether malicious or not); highly disciplined development practice; fewer flaws and less need for frequent patches and upgrades; and systems that do not overly rely on the constant vigilance of an inordinate number of skilled system administrators. People are clearly our most critical resource, but there is a serious shortage of computer professionals with a sense of history and theory, plus a strong grasp of the discipline that is necessary for sound development practice and sound operational practice. Hardware that enhances software security and reliability can also be very beneficial. Although certain mainstream processors have some security-related functionality that could approach the capabilities of the Multics hardware noted above, little of that functionality is actually used for trustworthiness. Some hardware aids are also clearly desirable, such as special-purpose co-processors in digital commerce, cryptography, and key management. Finally, demonstrations of how techniques for high assurance can realistically be incorporated into mainstream software developments would be extremely valuable. Our CHATS project is addressing all these directions.

6. Conclusions

The task of designing and implementing assuredly trustworthy systems and networks is inherently complex. It requires great diligence, effort, experience, and — above all — awareness of past mistakes and a commitment to avoiding them. Obliviousness to the past is a high-probability path to untrustworthy systems. We hope that our DARPA

CHATS report will provide you with useful pointers to the past and the future. Please read the cited report [10], and share your thoughts with us on its relevance within your own environments.

It is evident that enormous benefits should result from system and network developments that are highly principled and that use inherently sound architectures and sensible software engineering practices. This is not a new concept, although it seems to be seldom practiced. One enormous benefit therefrom would be the attainment of predictably seamless composability of systems (or perhaps analytically identifiable interactions) and starkly subsetted systems that avoid the usual plethora of exploitable vulnerabilities. This can entail compositions of demonstrably trustworthy subsystems, or in some cases compositions of less trustworthy subsystems whereby it is possible to explicitly demonstrate the attainability of greater trustworthiness (as discussed in [10]).

References

[1] H. Chen and D. Wagner. MOPS: An infrastructure for examining security properties of software. In *Ninth ACM Conference on Computer and Communications Security*, Washington, D.C., November 2002. ACM.

[2] H. Chen, D. Wagner, and D. Dean. Setuid demystified. In *Proceedings of the 11th USENIX Security 2002*, San Francisco, California, August 2002. USENIX.

[3] 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.

[4] 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.

[5] P.A. Karger and R.R. Schell. Multics security evaluation: Vulnerability analysis. In *Proceedings of the 18th Annual Computer Security Applications Conference (ACSAC), Classic Papers section*, Las Vegas, Nevada, December 2002. Originally available as U.S. Air Force report ESD-TR-74-193, Vol. II, Hanscomb Air Force Base, Massachusetts.

[6] P.A. Karger and R.R. Schell. Thirty years later: Lessons from the Multics security evaluation. In *Proceedings of the 18th Annual Computer Security Applications Conference (ACSAC), Classic Papers section*, Las Vegas, Nevada, December 2002. http://www.acsac.org/ .

[7] T.F. Lunt, R.R. Schell, W.R. Shockley, M. Heckman, and D. Warren. A near-term design for the SeaView multilevel database system. In *Proceedings of the 1988 Symposium on Security and Privacy*, pages 234–244, Oakland, California, April 1988. IEEE Computer Society.

[8] 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.

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

[10] P.G. Neumann. Principled assuredly trustworthy composable architectures. Technical report, Computer Science Laboratory, SRI International, Menlo Park, California, 2003. Final report, SRI Project 11459, June 28, 2003; see http://www.csl.sri.com/neumann/chats4.html; also chats4.ps and chats4.pdf.

[11] 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 ed., Report CSL-116.

[12] S. Owre and N. Shankar. Theory interpretations in PVS. Technical Report SRI-CSL-01-01, Computer Science Laboratory, SRI International, Menlo Park, CA, April 2001. http://www.csl.sri.com/~owre.

[13] 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).

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

[15] J.M. Rushby and B. Randell. A distributed secure system. *IEEE Computer*, 16(7):55–67, July 1983.

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