Software System Risk Management and Assurance*

Dr. Sharon K. Fletcher
Sandia National Laboratories
Albuquerque, NM, 87185-0449, USA

Ron Halbgewachs
Sandia National Laboratories
Albuquerque, NM, 87185-0877, USA

Roxana M. Jansma
Sandia National Laboratories
Albuquerque, NM, 87185-0484, USA

Martin D. Murphy
Sandia National Laboratories
Albuquerque, NM, 87185-0449, USA

Judy J. Lim
Sandia National Laboratories
Livermore, CA, 94551, USA

Dr. Gregory D. Wyss
Sandia National Laboratories
Albuquerque, NM, 87185-0747, USA

Abstract

Security, assurance, and risk management in software systems are viewed in terms of three historical generations, with significant paradigm shifts occurring in each generation. The software community is currently on the brink of the third generation, and needs advances in modeling, risk theory, tools, and assurance methods. The third generation is characterized by a broader, more integrative perspective on risk, and by modeling and measurement. This paper presents a third generation framework which demonstrates the viability of an integrative and quantitative approach.

Introduction

An historical look at software systems reveals a progression of thinking about protection and risk management. In this paper, three generations are defined. For each, we examine the prevalent views of risk, risk assessment, and risk mitigation. We also examine prevalent strategies for assurance.

Many gaps exist in current knowledge of how to manage and assess risks in software systems. This paper presents a fresh perspective which enables comprehensive risk-based design and evaluation of systems, spanning a range of surety concerns (including correctness and safety, in addition to traditional security concerns), and addressing multiple system aspects. We believe this to be a new and unique multidisciplinary approach which transcends both traditional security approaches and traditional risk analysis methods. It facilitates a risk analysis completely tailored to the system at hand, instantiating its threats, its barriers, and its needs for risk reduction.

This perspective considers risk states ("data integrity lost", "incorrect output", "system unavailable", etc.) and how such states might be reached given various starting conditions (normal operation, maintenance, etc.). Once the system is modeled from this viewpoint, barriers or risk-mitigators can be inserted into the model and their effectiveness can be estimated. The model must support a very broad interpretation of barriers, from software features, to physical protections, to operational procedures, to

* This work was sponsored by the U. S. Department of Energy under contract DE-AC04-94AL85000.

0-8186-7318-4/95 $4.00 © 1995 IEEE

software development methodologies, to design techniques. The modeling technique must allow system risk elements, such as barriers, to exert multiple influences throughout the system, so that we can deal realistically with complexities such as: conflicts between different surety objectives, secondary effects of elements, multiple uses of a single barrier, multiple barriers to a single risk, etc. The modeling technique must also allow one to define threat agents and to move them through the system in such a way that the model reacts to the progress of the agent. We believe our modeling technique provides a firm basis for such sophisticated analysis.

I. History of risk management for software systems

For purposes of discussion, we view the progression of thinking about risk management and assurance in terms of three "generations." These descriptions are meant to capture the prevalent views of the times, although there certainly might have been pioneers who were ahead of the times.

The first generation (rated systems)

The first generation of risk management was compliance-oriented, and required buy-in to a predefined set of risks which was assumed to apply to all systems. Risk concerns revolved around certain aspects of "CIA" (confidentiality, integrity, and availability). These views evolved in an environment characterized by mainframe computers and protection of classified information.

Not only was the set of risks fixed, but the mitigation strategies were dictated. These strategies included:

- access controls at the system and file levels (e.g., passwords, locks, MAC, and DAC)
- encryption for network transmissions
- disaster recovery planning.

There were several levels of systems defined, where a higher level meant more of this security model was implemented and/or it was implemented with more rigor for correctness.

In this generation, risk assessment was, for the most part, missing. Of course, some form of initial risk assessment occurred which defined the CIA set of risks for all systems. Beyond that, system-specific risk assessment usually included only site-specific disaster recovery concerns, and determining which security level applied. A system either did (compliant) or did not (non-compliant) implement the required strategies. Moreover, these strategies were based on stand-alone architectures where the security imposed on one system had little or no impact on the security for other systems. There was little leeway for customized solutions, never mind optimal solutions.

While restrictive, this approach succeeded in its environment.

The second generation (protecting assets)

Two things developed in the second generation: tool support for risk assessment, and a questioning of the universal applicability of the first generation view of risk management.

Risk assessment tools for software systems have appeared on the market in recent years. However, many of these are simply computerized checklists which measure compliance with the first generation's prescribed risk mitigators. Others are more aligned with the second generation view described below; however, many of these take the assets-protection viewpoint to an extreme and focus on computing Annualized Loss Expectancy, converting all assets to dollar equivalents.

The advent of, first, networks, and then, distributed processing on those networks, was very problematic for the first generation risk mitigation approach. The techniques that had been adopted did not easily extend into these more modern environments. Also at work was concern that the first generation CIA risk model simply did not fit all applications. A need was

felt for system-specific risk assessment. Other fields, such as nuclear power, weapons, and aviation, were taking a systems view and using analytical risk analyses; their success provided encouragement for a risk-assessment approach. As a result of all this, a new view of risk has emerged for software systems, and NIST played a prominent role in prompting this view to coalesce. [1] This new view is based on the following system components:

- vulnerabilities
- threats: active, passive
- assets: data, hardware, software
- impacts: disclosure, destruction, modification, unavailability
- types of mitigation: avoid, transfer, reduce threat, reduce vulnerability, reduce impact, detect & respond, recover

This emerging view of risk says "A threat is realized through a vulnerability, which impacts an asset." This more general view of risk, to be applied on a system-by-system basis, represents a major advance. Independently, Parker [2] has expanded and renamed (to "levels of abstraction") the assets list by breaking software into applications and operating systems, and adding users. He has also evolved CIA into a list of "security attributes" consisting of confidentiality, authenticity, integrity, utility, availability, and possession.

Little progress seems to have been made beyond these definitions, though, and this is due to two major roadblocks. First, the software community doesn't know how to measure the risk mitigation achieved by a design, and thus, how to draw any conclusions about assurance. Second, the community lacks a coherent framework for integrating assessment of the various aspects of security and safety and dependability, and therefore cannot easily assess the tradeoffs among these surety domains.

Positive aspects of the second generation's view of risk management include recognition of both passive and active threats, and recognition of a range of mitigation strategies. However, there are still limitations to this view. It is limiting because the concepts of impacts and assets don't encompass all we should care about. This view implies the system is operating properly to begin with, and we need only prevent the threats from being realized. This is still a fairly static view of systems; it does not lead the analyst to consider the full range of system lifecycle activities and states. And, while passive threats are acknowledged, they have not been fully explored.

In parallel with the second generation activities described above, progress is also being made on many fronts which contribute to higher integrity software systems. These include general software development methodologies, software system safety approaches, testing methodologies, access control models (e.g., role-based, constrained data items), domain specific architectures, requirements elicitation, etc. A general risk mitigation framework needs to be able to factor in the mitigating potential of these sorts of things as well.

The third generation (managing risk)

The third generation of risk management will be characterized by:

- A fundamental change of perspective to one which more fully facilitates total risk management
- Emphasis on correct system operation through appropriate levels of utility, access control, integrity, availability, and safety
- Consideration of actual threats, inherent vulnerabilities, and the feasibility and cost/benefit of safeguards as the basis for making system decisions.

This generation will have a dynamic, whole system, whole lifecycle perspective: build the right thing, build it well, and protect it appropriately. This is what we really care about.

Providing a viable framework is key; we must have a useful underlying perspective on risk assessment and risk management within which to work. It is imperative that enough effort be devoted to deriving a good framework, for this is the foundation, the view into the

problem space, that colors how we are able to see solutions. For example, while a narrow view of sabotage might focus on virus protection in an operational system, a whole lifecycle view will encompass protection throughout design, implementation, delivery, and maintenance. And while a narrow view of network security might focus on encrypting communications, a whole system view will explore whether network nodes have compatible security policies and whether they exchange sufficient security information to uphold the policies. And while a narrow view of integrity might address mechanisms within a properly operating database, a dynamic view will also look at shutdown-startup synchronization issues.

The framework should not reduce the problem to one of protecting assets. This is simply too narrow. How would safety be assessed? How would credit be given for thorough design practices? How would the risk of running the software in unplanned environments be recognized? How would one balance competing desires for availability and protection? These are important considerations if we wish to build the right system, build it well, and protect it. The framework especially should not reduce the problem to mathematics on apples and oranges. Real risks in real systems really do not reduce to Annualized Loss Expectancy dollars, and a single risk number is of little use anyway in improving the system.

The only alternative to moving into the third generation is to try to force a total risk management role onto the second generation perspective. This is kind of like trying to view integrity cast into the mold of confidentiality (an approach that has been suggested, but which has not met with a great deal of, or even a little, success). It is kludging new ideas into a paradigm that is too narrow to do them justice. There is danger in trying to saw with a hammer; the time comes to invent a more relevant tool.

II. Assurance Strategies In The Three Generations

First generation

The first generation made assurance straightforward for the consumer: buy rated products. It was also straightforward for the vendor: get rated. The picture was compliance-oriented.

Second generation

For many the answer's the same: buy rated products. This may still work where a first generation environment holds. A major problem being encountered, however, is the difficulty of composing rated products into today's more complex systems, which, if it can even be done, usually results in overkill solutions at unacceptably high cost.

But developers are increasingly rejecting the underlying mindset that simply complying with orders actually provides the needed surety. The second generation risk model represents a positive trend toward assessing a system's actual surety needs. Unfortunately, it does not have a sanctioned assurance technique, at least not in the US The Federal Criteria [3] may provide a step in the direction of customized risk mitigation, but it's still within the rated product approach.

Many emerging methodologies which contribute to high integrity systems are not addressed by either the risk model or the Federal Criteria. As long as there is not an assurance technique that credits good practices, developers will unfortunately sacrifice doing things right in order to apply scarce resources to doing things that are measured.

Third generation

The third generation assurance mindset must migrate from component assurance to system assurance. It must be aligned with a system

view that is more than the sum of a set of surety tools. It must encourage integrating the assessment of dependability, security, safety, and cross fertilization between approaches for richer solutions. It must deal with interdependencies, system dynamics, and with the whole lifecycle. It must allow for application of cost-effective measures commensurate with actual requirements. And it must allow for solutions which represent real surety as opposed to compliance. (Situations can be cited where strict compliance is not even functional, much less secure. For example, allowing only DES encryption leaves no functional way to distribute keys.)

Third generation assurance will not be easy. It must begin by developing a common language for surety requirements, that is, an ability to specify degrees of risk-mitigation required in various areas for a system. This gives a developer the ability to tailor a surety model for the system (or at least select from an available set of models). Next, the developer needs assistance in selecting risk-mitigating techniques, and understanding their effectiveness, and dependencies and interactions. This leads to a system design, and perhaps also to development and operational methodologies. Finally, there needs to be some accountability in applying the selected techniques. This final step is the assurance that the expected benefit is actually realized.

III. An Example Of A Third Generation Risk Framework

We envision a risk management framework where surety requirements are determined in terms of perceived risk and desired risk reduction, within the context of a **risk identification matrix**. Then, the effectiveness and interaction of risk mitigators is explored, within the context of a **system risk model**. Technology providers or others may populate a **risk mitigators matrix** with some nominal assessments of the risk mitigating potential of the technologies. However, moving the mitigators into the system risk model instantiates them in a particular setting and links them with other elements in the system, in order to achieve a realistic picture of the effect of using that barrier in the particular system at hand. The result of all this is a **risk assessment** and a **risk management strategy** for the system. This, in addition to auditing/documenting the actual application of chosen technologies would provide further evidence for **assurance**.

This vision of software system total risk management is explored in more detail in this section. A research project at Sandia National Laboratories is developing the concepts put forth here. [4,5] These concepts form what we feel is a viable risk management framework.

The components of our vision are:

- risk identification matrix
- risk mitigators matrix
- system risk model
- process:
    - building the model
    - barrier analysis
    - threat analysis
    - analysis engine
        - risk evaluation
        - cost/benefit evaluation

The risk identification matrix provides a taxonomy of risk sources for software-based systems. The rows of the matrix represent surety objectives, and the columns represent aspects of a system which might give rise to risks. The cells of the matrix contain sources of risk. The intent is to populate each cell to eventually contain all possible relevant sources of risk for any system, arranged as pieces of a taxonomy.

The matrix is read:
"There is a [surety objective] risk relative to [system aspect] due to [risk]."

Examples:

* "There is an [access control] risk relative to [system composition : network] due to [passwords exposed on network]."

* "There is an [integrity] risk relative to [information] due to [processing error]."
* "There is a [utility] risk relative to [state changes : shutdown] due to [shutdown-startup not synchronized]."
* "There is an [availability] risk relative to [processes] due to [system overload]."
* "There is a [safety] risk relative to [interfaces] due to [unchecked input]."

Although the traditional impacts-assets are accommodated within this framework, it is much broader, giving rise to exploration of system dynamics (state changes), architecture choices (composition), and correct operation (utility). For a particular system undergoing analysis, the risk identification matrix will be pruned by the analyst to contain and prioritize only those risks of sufficient consequence and likelihood that they need to be mitigated. Consequences that should be considered include mission-related, political/social, health & safety, environmental, and regulatory/legal.

The risk mitigators matrix has the same format as the risk identification matrix, but contains mitigators corresponding to risks. The intent is that the mitigators not be limited to hardware and software technologies, but include rules and procedures, design and development practices, and cover the lifecycle spectrum. Thus credit can be given for using a proven real-time design architecture, for using a highly rated software development methodology, for a trusted path delivery mechanism, for a fail-safe design, etc.

A straightforward but simplified way of diagramming the system risk model is illustrated in Figure 2.

Elements of the model are system states or events, represented by circles; transitions, represented by lines between circles; and risk

	Information	Processes/ Transactions	Composition hw-sw-nw-usr	State Changes op-mt-sh-ae	Interfaces
Access Control			passwords exposed on network		
Integrity	- intruder alters - processing error - user alters				
Utility				shutdown-startup not synchronized	
Availability		system overload			
Safety					unchecked input

hw = hardware
sw = software
nw = network
usr = user/operator

op = operational
mt = maintenance
sh = shutdown
ae = abnormal event

Figure 1. Risk identification matrix

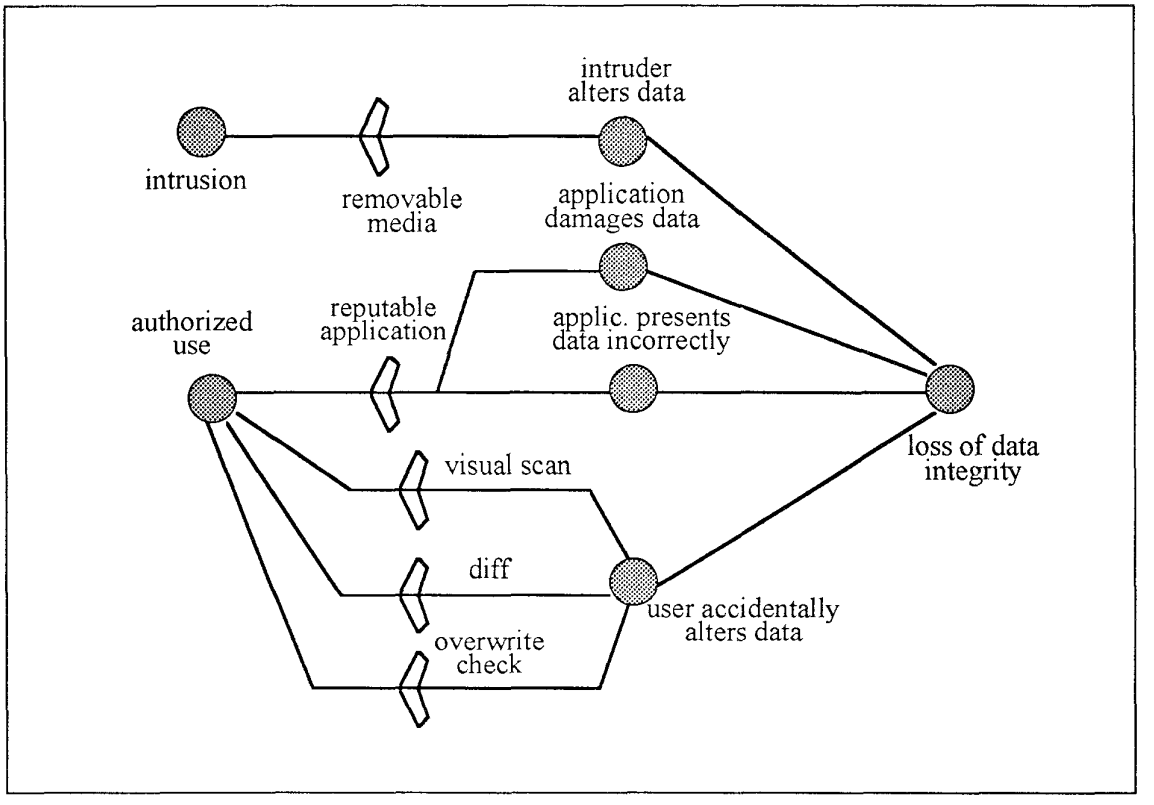

Figure 2. System risk graph

mitigators, represented by the barrier symbol along transitions. This type of modeling has been applied successfully in other fields, but is not commonly used yet in the software field. The example illustrates how one mitigator (use a reputable application) can mitigate two transitions, and how a variety of mitigators (visual scan, diff, overwrite check) can be considered for mitigating a single transition.

Broadly, the process consists of first arriving at a system risk model, through use of the risk identification matrix and risk mitigators matrix, then performing threat and barrier analysis within the model, and then running an analysis engine to compute risk along the various paths.

In the example shown in the figure, the risk being explored is loss of information integrity (in a spreadsheet, for example) in an operational system. This would be only a part of some system's total risk model. Presumably, the analyst has deemed this high enough in likelihood and consequence to warrant this level of breakdown and analysis.

Barrier analysis is instantiation and refinement of a risk mitigator's ability to mitigate system specific risks. Several characteristics of risk mitigators are considered, including how much technology vs. how much rules-and-procedures (rap) are involved, perceived strength, cost to implement, ease of use, outside dependencies. In the example, preventing the user from accidentally altering data by requiring a visual scan is entirely rap, not very strong, and hard to use. Providing the user a "diff" tool is a stronger technology, easier

to apply, and still has some rap component (the user must remember to use it). Providing some sort of automatic overwrite check is stronger yet, has even less of a rap component (the user must still respond appropriately), but may be implemented in such a way as to have a high annoyance factor which may cause the user to ultimately defeat it. All these considerations lead to an estimate of each barrier's ability to reduce the likelihood of transition to the undesired next state.

Threat agents may be active or passive. Active threat agents may have characteristics such as motivation, skills, knowledge, time and other resources. They are willing to incur some amount of cost and risk, in order to gain the perceived value of their target. Passive threats, representing unintentional faults in the system, may have other characteristics. In either case, as a threat unfolds into the system, the agent's characteristics and the system's surety elements may both be altered by the interactions.

The analysis engine must combine transition probabilities, threat estimates, barrier estimates, and risk reduction requirements to yield information on remaining risk. Uncertainty analysis must accompany the calculations, so that the engine can target highly uncertain calculations for refinement. Furthermore, the corresponding costs - monetary and social - that come with the reduction in risk must be factored into the analysis to provide a complete framework for decision making. The appropriate mathematics is a current subject of investigation.

All the concepts described here are still evolving.

IV. Beyond the Framework - Better Surety Requirements

A natural extension to the work described in Section III is to provide for better expression of software surety requirements for a system. Today, the software community does not really have a commonly accepted language for expressing the functional requirements of a system; in fact, software requirements elicitation is an active research topic. [6,7] If we can barely do an adequate job with functional requirements, we are really lacking in our ability to express surety requirements. Unfortunately, this is true even for high consequence operations, such as in the nuclear power industry as it become more software-dependent.

In light of our risk management framework, surety requirements can be viewed as statements of the need for risk reduction. An expression representing total system risk is one of the natural outputs of the analysis engine previously discussed. It should be possible, then, to derive requirements as a mathematical complement of the risk expression. And, since the system risk model is designed to accommodate interactions and dependencies among risk-related elements of the system, we could, theoretically, obtain a more optimal set of requirements in which redundancy has been reduced by running an initial set of requirements through the analysis engine. Finally, when coupled with cost-benefit analysis, the resultant risk expression offers the opportunity to achieve the most risk reduction for the resources allocated. Thus, it may be possible to adjust the requirement set to reflect the desired balance between various surety objectives.

Variations of this approach have been used to optimize the design or upgrade of control systems and safeguards systems. A fundamental property of this type of derivation is that the requirements will be complete if and only if the risk expression is complete. That is, the resultant requirements will be only as good as the risk model. While these ideas seem to hold promise as a methodology for expressing surety requirements, much work needs to be done.

Summary

This paper has argued for the need to replace the traditional views of computer security and risk management with one which is broad, integrated, and useful for managing risk throughout the life of a software system. The evolution of thought in these areas was

presented as a set of "generations" in order to emphasize the need for significant paradigm shifts. We are on the threshold of the third generation, which is more encompassing, integrated, and tailorable than previous generations. Third generation methodologies and tools need to be derived for the software community-at-large. Challenges for the third generation include developing a broad-perspective software system risk theory, developing effective tools, and re-defining assurance to be based on measurable risk reduction rather than on compliance.

This paper presents one possible third generation approach. This approach offers the ability to explore tradeoffs and interactions among various surety objectives and techniques. It brings system dynamics and lifecycle concerns into an up-front analysis, and produces a record of surety requirements and decisions. The approach can be used now in a qualitative sense, and can become more quantitative as the theory is refined and data are collected.

References

1. Proceedings of the 4th International Computer Security Risk Management Model Builders Workshop. August 6-8, 1991. Sponsored by NIST and the University of Maryland.

2. Parker, D., "Restating the Foundation of Information Security," Proceedings of the 14th National Computer Security Conference, October, 1991, Washington, DC.

3. Federal Criteria for Information Technology Security, Published by NIST and NSA.

4. Fletcher, S. K., "The Risk-Based Information System Design Paradigm," Proceedings of the IFIP SEC'94 Conference, May 23-27, 1994, Curacao, NA.

5. Jansma, R. M., et. al., "Risk-Based Assessment of the Surety of Information Systems," Proceedings of the 11th International Symposium on the Creation of Electronic Health Record Systems and Global Conference on Patient Cards, March 13-18, 1994, Orlando, FL.

6. Kang, K. C. and M. G. Christel, Issues in Requirements Elicitation, SEI-92-TR-012. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

7. Lubars, M., et. al., A Review of the State of the Practice in Requirements Modeling, MCC Technical Report Number RQ-169-92. Microelectronics and Computer Technology Corporation, Austin, TX.