Understanding and Managing Risk in Software Systems

Dr. Sharon K. Fletcher
Roxana Jansma, Judy Lim, Martin Murphy, Dr. Gregory Wyss

Corresponding Author:
Dr. Sharon K. Fletcher
Sandia National Laboratories
MS0777
P. O. Box 5800
Albuquerque, NM, 87185-0777, USA
Phone: 505/844-2251 // Fax: 505/844-9641 // email: skfletc@sandia.gov

Abstract

When software is used in safety-critical, security-critical, or mission-critical situations, it is imperative to understand and manage the risks involved. A risk assessment methodology and toolset have been developed which are specific to software systems. This paper describes the concepts of the methodology, with emphasis on the experience of defining the tools to support the methodology. Also presented are results of applying the methodology to two real software-based products: the software tools themselves, and a network firewall.

This work was supported by the United States Department of Energy under Contract DE-AC04-94AL85000.
DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.
Understanding and Managing Risk in Software Systems

Abstract

When software is used in safety-critical, security-critical, or mission-critical situations, it is imperative to understand and manage the risks involved. A risk assessment methodology and toolset have been developed which are specific to software systems. This paper describes the concepts of the methodology, with emphasis on the experience of designing a toolset to support the methodology. Also presented are results of applying the methodology to two real software-based products: the software toolset itself, and a network firewall.

Introduction

This paper relates a two-year experience in building and applying a new methodology for assessing the surety of software-based systems. Surety includes security, safety, dependability -- all the desirable attributes of a system that might be considered "ancillary" to the functional requirements. Our primary goal in this research was to develop a methodology for surety assessment of software-based systems. A secondary goal was to provide some automated tool support for the methodology.

We cast the problem in terms of surety risks and sought a framework that would enable software system design based on risk management. We identified several aspects of risk-based design that need to be supported by the methodology. First, a designer must be able to identify the risk issues for the system, over the entire lifecycle, and including issues of dynamics that arise when transitioning among operation, maintenance, and planned or unplanned shutdown states. Second, the designer must understand and document how a proposed design mitigates the surety risks. This becomes the basis not only for the design, but also for certification and for understanding the impact of future changes. Finally, the designer needs a tool for exploring tradeoffs among design alternatives. It is not unusual to have competing surety objectives, or to have mitigators that affect more than one objective (positively or negatively), leading to complex tradeoff considerations.

Perspective & Concepts

Overview

This section describes the concepts and process of the overall methodology we have developed, although some pieces are not yet fully developed.

We provide the analyst a framework for defining perceived risk and desired risk reduction, within the context of a risk identification matrix. Then the effectiveness and interaction of risk mitigators (barriers) is explored, within the context of a system risk model. A risk mitigators matrix provides information on potential barriers and their nominal effectiveness. However, by moving the mitigators into the system risk model, the analyst instantiates them in a particular setting and links them with other elements in the system to achieve a realistic picture of the effect of
using that barrier in the particular system at hand.

The system risk model provides a graphical depiction of potential system states and the barriers that can affect the probabilities of state transitions. But, more than that, it is also the formal description over which risk calculations can be defined. Thus, it is the heart of the analysis method. While such modeling is common in other fields using probabilistic risk assessment, we felt that software system designers and analysts might not be familiar with this kind of modeling, and might have difficulty constructing meaningful and complete models. The matrices were introduced as an aid to the model building.

The overall process, as we envision it, is for an analyst to build a system risk model, using the risk identification and risk mitigators matrices as guides and sources of information. Then the analyst performs a barrier analysis (which is the quantification and instantiation of the barrier into the model), and a threat analysis for each threat that is to be considered. Then an analysis engine is run to evaluate remaining risk in the system. The job of the analysis engine is to do appropriate computations on all quantified input from the analyst, and to return information on weaknesses in the system. If cost information is incorporated, then the analysis engine returns cost/benefit information as well.

The result of this process is a risk assessment and a risk management strategy for the system.

Detail

Of the concepts described above, all except threat analysis and cost/benefit evaluation have been developed to some degree of detail.

The risk identification matrix, Figure 1, 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. Ideally, we would like to populate the matrix with all possible relevant sources of risk for any system, arranged as pieces of a taxonomy. This would ensure breadth of analysis, by leading the analyst to consider all possible risks. The analyst could select which risks to address, and prune out the rest. In reality, we make it as complete as we can, and provide a mechanism for the analyst to prune and add to it.

The matrix is read:

“There is a [surety objective] risk relative to [system aspect] due to [risk source].”
**Table: Risk Identification Matrix**

<table>
<thead>
<tr>
<th></th>
<th>Information</th>
<th>Processes/Transactions</th>
<th>System Composition</th>
<th>State Changes</th>
<th>Interfaces</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>hw-sw-nw-usr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Access Control</td>
<td></td>
<td></td>
<td>passwords exposed</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>on network</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integrity</td>
<td>- intruder alters</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- processing error</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- user alters</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Utility</td>
<td></td>
<td></td>
<td></td>
<td>shutdown-startup</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>not synchronized</td>
<td></td>
</tr>
<tr>
<td>Availability</td>
<td></td>
<td></td>
<td>system overload</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Safety</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>unchecked input</td>
</tr>
</tbody>
</table>

| hw = hardware    | op = operational  |
| sw = software    | mt = maintenance  |
| nw = network     | sh = shutdown     |
| usr = user/operator | ae = abnormal event |

**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]."

Others have adopted a perspective based on protection of assets, typically defining assets as hardware, software, and information. [1] Then they have considered what types of impacts to assets might occur, such as destruction, modification, and disclosure, and how such impacts might be mitigated, such as threat reduction, vulnerability reduction, detection and recovery, etc. This is a useful line of thinking for moving away from the computer security compliance-based world into a risk-based world. However, we wish to address a broader problem.

Although the traditional impacts-assets are accommodated within our framework, it is much broader, giving rise to exploration of architecture.
choices (composition), system dynamics (state changes), system environment (interfaces), and correct operation (utility). The intended purpose of the matrix is to guide the analyst's thinking into all relevant areas of risk and to suggest, but not limit, risks that should be considered. In our view, it is not important for the categories to be non-overlapping; what is important is to stimulate thinking about all aspects of the system and all surety objectives. The "composition" aspect, for instance, includes risks that would be inherent in certain choices of platforms, network, and communications architectures. "State changes" includes risks that might be present in maintenance procedures and in cases of abnormal system shutdown, etc. The "interfaces" aspect comprises the context in which the system actually functions. The "utility" surety objective implies correctness and fitness-for-purpose of the system. We envision arranging the risks in the matrix into pieces of a taxonomy, so that the analyst can consider a high level description of a risk, and then delve into more specific and detailed sources of that risk. Thus analysis can be carried out at very low levels of detail in areas of great concern, and higher levels in other areas, all rolled up into a single model of the total system.

For a particular system undergoing analysis, the risk identification matrix will be both pruned and extended 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 consequences form a third dimension on the matrix, not shown in the figure.

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. When available, nominal barrier analysis metrics (described below) accompany each mitigator. This gives the analyst a starting place for instantiating the barrier into the system risk model. A metric always consists of a quantified value plus a statement about the uncertainty of that value. Our work currently focuses on quantifying existing risk mitigators, not necessarily inventing new ones. All the good work that is going on everywhere in security, software safety, and software development processes, will be sources for populating this matrix.
The heart of the analysis technique is the system risk model. The matrices are used by the analyst as guidance for constructing the model. One way of diagramming the system risk model is illustrated in Figure 2.

Elements of the model are system states or events, represented here by circles; transitions, represented by lines between circles; and risk mitigators, represented by the barrier symbol along transitions. The figure 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. We use this visual representation for clarity; however, the interactions of elements can actually be more complex than they may appear in this simplistic example. Mathematically, the probability of reaching any state is a function of all previous states that can transition to this state, and all barriers that can influence the transitions (regardless of whether they are drawn on the path).

In the example shown in the figure, the risk being explored is loss of information integrity in a database, 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 the estimate of a risk mitigator’s ability to mitigate transitions to risk states. 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. Once again, each metric is accompanied by a statement of uncertainty.

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 invoke 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 an undesired next state. Figure 3 illustrates a coarse quantification of barrier characteristics, without the uncertainty values.

The concept of threat agents is not yet developed, but it is envisioned that agents will have definable characteristics as well as influences on transition probabilities. Useful analysis can be carried out without defining specific threat agents, but when this concept is added, more sophisticated analyses will be possible.

The analysis engine combines transition probabilities, barrier and threat estimates, and risk reduction requirements to yield information on remaining risk. In particular, the engine identifies paths in the risk model where risk is still too high. Uncertainty analysis accompanies the calculations, so that the engine can also target highly uncertain calculations for refinement. The analysis engine is currently under development. The math or logic for folding barrier and threat influences into transition probabilities, and for combining uncertainties, is under study.

Iterative refinement is basic to the way in which an analysis should be carried out. The analyst should first work in the context of a high level system risk model, input estimates, run the analysis engine, and examine the results. High risk paths can then be strengthened with
additional barriers, or can be broken down into more detail. Highly uncertain paths that are determined to be of sufficiently high consequence call for better estimates, and they may also benefit from refinement. Once the highest level risks have been adequately addressed, the analyst may wish to incorporate additional risks into the model and continue the analysis. The analyst will be able to see the total impact of old and new barriers on all risks.

Case Studies
The methodology has been experimentally applied to two real cases. First, we considered that we are aiming to provide the software community a tool for risk management. Surely there are risks in trusting such a tool -- risk to the user's mission, and perhaps regulatory or social (embarrassment) risks as well -- should the tool mislead the user about system risks. Because we want the tool to be a sound and useful product, we decided to subject it to a risk analysis.

Second, we selected a network firewall which has been designed and implemented, but which has not undergone a rigorous or systematic analysis. We wanted to see 1) if our method is systematic and helpful enough to "discover" the risks that have already been recognized in the design, and 2) if we could improve the firewall by uncovering additional risks.

(These analyses are underway as this draft is being written. Results and lessons learned will be reported at the conference.)

Automating the Process via Software Tools
The methodology we are developing is necessarily rich and complex, in order to facilitate realistic risk analyses. To use such a methodology without good tool support would be nearly impossible. There must be software tools to aid in constructing the matrices and graphs and to automate the calculations; and there should be libraries which capture knowledge for reuse. However, the methodology itself is in its infancy, so these tools must be able to grow and change as the methodology evolves.

Thus we felt it was important to design the toolset for flexibility. In particular, we wanted to avoid being locked into any particular mathematics or logic for computing risk. We wanted to be able to experimentally vary how barrier and threat metrics influence transition probabilities, and allow for them to interact with each other. For example, a threat agent’s resource quotas or risk tolerance might get “used up” as barriers are traversed.

In developing the methodology, we found ourselves with a multitude of concepts, but without a clear understanding of how they relate to each other. We had concepts of risk states, events, risk sources, barriers, influences, consequences, threat agents, scenarios, matrices, graphs, taxonomies, and so on. We realized that we could not build flexible tools (maybe not even viable tools) until we imposed some structure on the concepts.

For these reasons, we decided to try to express the concepts, and the process of performing a risk analysis, with some type of formal semantics. This would
not only force clearer thinking and more precise definition, but it would also suggest a flexible architecture for the tools. This activity turned out to be highly beneficial.

**Using Semantics to Express Concepts & Process**

At the beginning of this exercise in semantics, we had developed ideas and terminology in three areas: 1) the risk identification and mitigation matrices, 2) the system risk graph, and 3) the computations of the analysis engine. We were struggling with how to pull these together into one coherent methodology, and with the question of whether we were calling the same things by different names.

We began by expressing concepts and relationships between concepts in structured English sentences patterned after the Natural language Information Analysis Methodology (NIAM). [2] We found this exercise difficult, and it had the desired effect! It forced increased clarity of thought and illuminated where we had similar concepts with different names. By resolving the naming differences, we were able to see how to make the transitions between the three disjoint parts of our methodology.

For example, a risk source in the risk identification matrix becomes a state node in the system risk model. Likewise, a piece of taxonomy from the matrix becomes a subgraph. Further, the corresponding matrix row and column headings sometimes suggest what should precede and follow the node in the model. We began to see how we might automate much of the work of deriving the model. Additionally, we saw an architecture for the software tool that had a clean separation of data elements and processes acting on those elements. With this architecture, it would be possible to swap out processing algorithms to try different logic and math. It would also be possible to later extend concepts and the processes to deal with them. Some of the needed processes were implied by the semantics (e.g., convert a risk source into a node in the model), and others we could define based on the data structures that the semantics showed us were available to be acted upon.

**Conclusion**

We have developed a risk assessment methodology for software systems, which we hope will assist the software community in developing surety-critical software. The methodology is still in its infancy, but it is already supported by tools which make it viable. We have shared the methodology, the results of its application, and the experience of its development.

**References**
