TOWARDS A SCIENCE OF SURVIVABILITY:
A Research Agenda and a Specific Method

Jeannette M. Wing
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

24 July 2000

I propose four broadly stated agenda items for providing a foundation for understanding survivability: what it means and how we can design systems with it in mind. I also briefly describe a specific method and tool that we are using that speaks to these general research goals.

Four Research Agenda Items

Let's start with an assertion. By definition a survivable system must tolerate faults: both accidental and malicious; both expected and unexpected.

1. Revisit results from fault-tolerant research in light of security, and vice versa.

Today people design systems to be fault-tolerant or secure, but not both. They assume well-behaved users, but unreliable system components, or they assume reliable system components but ill-intentioned users. The designer of a survivable system must assume system components are unreliable and users untrustworthy.

Thus, techniques for ensuring survivability should be inspired by those for making systems more fault-tolerant (to handle accidental faults) and by those for making systems more secure (to handle malicious faults). "Inspired by" does not mean "applied directly to." The tricky part is that a given technique may make assumptions that do not hold in a context broader than originally imagined. For example, many of the results from the fault-tolerant community (including the use of redundant hardware, database transactions, and Byzantine agreement and other distributed consensus protocols) hold under the assumption that failures occur independently. Some techniques are devised to ensure that this assumption will hold. (For example, strict two-phase locking guarantees no cascading aborts.) In practice, this assumption may not hold; the failure of a system component may be triggered only in the event of the failure of another. If so, then an attacker can exploit this failure dependency to his or her advantage. These simple observations suggest:

- We need to understand better when a given technique for increasing fault-tolerance (e.g., use of replication of system components to increase reliability and availability) is valid when also trying to increase a system's security (e.g., use of cryptography, authentication protocols, access control mechanisms, firewalls); and vice versa.

- We need to understand how the different techniques interact with

each other. A given technique's set of assumptions may be violated by the effects of another. Or, a given technique's set of guarantees may be subsumed by another; we can avoid unnecessary redundancy in this case. Or, the same technique (e.g., diversity of system components) used both to address fault-tolerance and to address security in isolation may no longer work when addressing both at the same time.

To come to some understanding of when one technique applies or not, and how different techniques work together,

- We need a class of models for describing a system's behavior that is rich enough so we can determine the effects of the interaction between different techniques.

- We need a logic (a specification language and verification method) that allows us to specify a system's behavior, to assert system properties, and to verify that a given system model satisfies or violates these properties.

2. Devise models to handle both expected and unexpected failures.

Ideally, in designing a system to be survivable, we would identify all potential conditions of failure and guard against their occurrence, mitigate their effects if a failure occurs, and if necessary, repair and restore the system to normal operation. In practice, however, it is impossible to predict all failure conditions; we must design a system with in mind that a system's environment may be unpredictable, perhaps due to natural disasters or humans being in the loop. (On the one hand, humans make mistakes, and on the other, they cleverly exploit system vulnerabilities.) Thus,

- Our models and logics must be rich enough to accommodate unpredictable environmental behavior.

3. Devise models to handle finer-grained behavior.

There are many dimensions of granularity we could consider. Here are just a few to think about.

First, rather than think of fault-tolerance and security, and hence survivability, as boolean properties, we should think in terms of degrees of reliability and degrees of trustworthiness.

Even for predictable failures, not all are expected with equal likelihood. For example, we may assume that the links between the Federal Reserve Banks are highly reliable (e.g., because of redundancy), but those between ATM machines and banks are less reliable (because it is not worth the cost). A model should be rich enough to let us model degrees of reliability, e.g., measured in terms of probability of failure.

Similarly, even for trusted components, some may be trusted more than others. For example, we may assume that the system administrator for the bank's database server is trustworthy, but that all ATM users are potentially not. A model should be expressive enough to let us specify fine-grained access control, along many dimensions including application, history, context, and resource usage. We should be able

to state context-sensitive security policies such as "When the editor is controlled by user A, it can access any file; otherwise it can only access /tmp" or resource-sensitive policies such as "This applet can only send 1KB/minute."

Second, the fault models for a survivable system should consider degrees of severity of a fault and degrees of maliciousness. For example, the fail-stop model from the fault-tolerant computing community is inappropriate for understanding survivability; any malicious fault that is not self-contained (e.g., a virus or a distributed denial of service attack) falls outside of this model. Also, the Byzantine fault model from the fault-tolerant computing community seems too general for handling degrees of malicious behavior, both in the sense that not all users are malicious and that some malicious acts have graver consequences than others.

Thus,

- We need models that support finer-grained analysis along multiple dimensions. They should at least (1) exploit information about why a failure has occurred and who or what caused it; (2) allow direct cost-benefit analysis with respect to tolerating faults and intrusions; and (3) allow dependency analysis between failures.

4. Revisit fault-tolerance, security, and hence survivability, in light of the Internet computing environment.

A survivable system must be both fault-tolerant and secure. Neither of those properties is new. As argued above, considering them together in a system design is the new challenge.

But what is brand new? The way in which systems are constructed today. Systems are not closed, static entities. Open system architectures make it more challenging to design and analyze a system to be fault-tolerant, secure, and hence survivable. Ditto for software applications that use mobile code (e.g., Java, ActiveX), the programming paradigm of the Internet.

Also, the increasing interdependencies of system infrastructures create more opportunities for points of failure and targets of attack. So the scale (size and complexity) of the today's problem has outgrown yesterday's solutions.

Given the openness of systems today, in terms of design, use, and access:

- We need new models of computing to grapple with systems with no boundaries, with greater anonymity of users, and with scales in size and time of the Internet. These new models will likely lead to new distributed algorithms, new distributed system architectures, etc. These models might benefit from thinking in terms of biological systems metaphors (e.g., immunology, epidemiology, and ecology), and from applying rich mathematical theories (e.g., game theory, Markov Decision Processes, decision analysis, econometrics).

Finally, Ellison et al. argue that it does not make sense to talk about the survivability of a system without putting it in the context of the "mission", i.e., the end service, that the system provides,

e.g., electricity, telephony, or health-care [Ellison et al.97]. They further argue that one identifies a mission's "essential services" and design a system to ensure these are sustained in the presence of faults. Thus,

- We should understand more formally what the role of the mission (and its essential services) plays in the notion of survivability. Is it fundamental in some way, and if so, how? Or is it merely useful as a design criterion?

A Specific Example

Towards this broad research agenda, my colleagues and I defined a two-phased method for analyzing the survivability of a networked system [JWLL00], and Somesh Jha has implemented a tool, Trishul, to support it.

Phase One

We build a scenario graph representing the sequences of states and events that lead to final states that satisfy a given property.

Phase Two

We annotate a subset of the scenario graph's edges (events) with probabilities, which can be symbolic or numeric, such that we can compute the cost of a given function, e.g., the worst-case reliability metric of a given property.

For Phase One, we modified an off-the-shelf model checker, NuSMV, to produce the scenario graph automatically. Suppose we want to express the property that "It is not possible for a node N to reach a faulty state if the network starts from one of the initial states." Letting "fault" represent the property that node N is in a faulty state, we can express the property as a CTL formula as "AG (~fault)". Feeding this assertion into a model checker, which produces counterexamples (i.e., sequences of states for which the negation of the property holds in the final state), gives us the scenarios for which N reaches an undesirable (faulty) state.

For Phase Two, we first construct a Bayesian network, which allows us to model dependent events, for some (not necessarily) all fault events. We use these conditional probabilities to annotate the scenario graph produced by Phase One. We are left then with a state transition system that has both nondeterministic states (states with unlabeled outgoing edges) and probabilistic states (states with outgoing edges labeled with probabilities). Interpreting the nondeterministic transitions as the environment making a choice (thus modeling some degree of unpredictability), for computing a worst case reliability metric, we use a policy iteration method similar to that used for optimal control of Markov Decision Processes (MDPs). This method determines the worst case scenarios for a given property.

Currently we are reimplementing our support for Phase One using a different model checker since for some of the industrial-sized case studies, we cannot handle a large enough state space with NuSMV. We also plan to generalize the analysis we do in Phase Two, allowing

different cost functions and allowing other analyses besides worst-case reliability. We also plan to hook up our new tool to a linear programming package to exploit the power of off-the-shelf automated analyses of MDPs.

References

[Ellison et al. 97] Ellison et al., "Survivable Network Systems: An Emerging Discipline," Carnegie Mellon University/Software Engineering Institute, CMU/SEI-97-TR-013, November 1997, revised May 1999.

[JWLL00] Somesh Jha, Jeannette M. Wing, Richard Linger, and Tom Longstaff, ``Analyzing Survivability Properties of Specifications of Networks,'' in Proceedings of the International Conference on Dependable Systems and Networks, Workshop on Dependability Despite Malicious Faults, New York City, NY, June 25-28, 2000, pp. 613-622.