VALIDATION OF A SURVIVABLE PUBLISH-SUBSCRIBE SYSTEM

Sankalp Singh[†], Adnan Agbaria[†], Fabrice Stevens[‡], Tod Courtney[†], John F. Meyer[††], William H. Sanders[†], Partha Pal[‡‡]

[†]University of Illinois at Urbana-Champaign, USA.{sankalps, adnan, tod, whs}@crhc.uiuc.edu
[‡]France Telecom, France. fabrice.stevens@francetelecom.com
[††]University of Michigan, USA. jfm@eecs.umich.edu
[‡‡]BBN Technologies, Cambridge, MA, USA. ppal@bbn.com

Abstract: *We describe, with respect to high-level survivability requirements, the validation of a survivable publish-subscribe system that is under development. We use a top-down approach that methodically breaks the task of validation into manageable tasks, and for each task, applies techniques best suited to its accomplishment. These efforts can be largely independent and use a variety of validation techniques, and the results, which complement and supplement each other, are seamlessly integrated to provide a convincing assurance argument. We also demonstrate the use of model-based validation techniques, as a part of the overall validation procedure, to guide the system's design by exploring different configurations and evaluating trade-offs.*

Keywords: Quantitative Validation, Security Verification, Information Assurance, Probabilistic Modeling, Intrusion Tolerance, Security

1 INTRODUCTION

The emergence of large distributed information systems to support nation-critical needs (e.g., electrical power distribution, telecommunications, military command and control, and health care) has spurred research into new system protection strategies that can produce a system whose critical function will *survive* in spite of hostile attacks, complex failures, or accidents [1]. These new strategies have drawn upon traditional approaches such as prevention technologies (e.g., cryptography, authentication, and firewalls), detection technologies (e.g., network sensors and host-based sensors), fault-tolerance technologies (e.g., agreement protocols), and the like, and combined them with newer technologies that enable dynamic defensive responses such as changing security policies or isolating suspected components. Successful strategies interleave these building blocks into a solution that remains cost-effective and manageable even when scaled to very large systems.

The validation of survivable systems, like their construction, must use an integrated approach. In addition to well-known security requirements such as confidentiality and integrity, a survivable system may also need to satisfy probabilistic constraints on its behavior. Such constraints impose different modeling and reasoning strategies. This is particularly true of survivable systems that make use of intrusion tolerance technologies [2, 3, 4], since by definition, intrusion tolerance is a probabilistically quantified property of the system. Moreover, since impairments may manifest themselves in implementation details, we must rely on focused testing to convince ourselves that the implementation is faithful to the modeled design. A successful validation approach must collect this disparate evidence into a single, seamlessly integrated assurance argument that is convincing to the accreditor.

Most traditional approaches to validation of security have been process-based, usually in the form of guidelines [5]. Goal-based evaluations, when attempted, have usually either been based on formal methods [6] and aimed to prove that certain security properties hold given a specified set of assumptions, or been informal, using teams of experts (often called "red teams," e.g., [7]) to try to compromise a system. Quantitative methods, in particular probabilistic modeling, have been receiving increasing attention as a mechanism to validate security. Any probabilistic model intended to validate a secure system would have to represent, among other things, the attacker's behavior. Since some of the vulnerabilities in an information system will be unknown at the time the system is designed (and modeled), the prediction of when and how an attacker may successfully intrude the system is a difficult, but critically important problem. Early work on probabilistic validation of secure systems was done by Littlewood et al. [8]. Their work was exploratory in nature and identified "effort" made by an attacker as an appropriate measure of the security of the system. Jonsson and Olovsson [9] attempted to build a quantitative model of attacker behavior using data from several experiments they conducted over a two-year period. They postulated that the process representing an attacker may be broken into multiple phases, each of which has an exponential time distribution. Attempts have been made to build

models that take into account behavior of the system as well as the attacker, and the uncertainties therein. Madan et al. [10] have used a semi-Markov model to evaluate the security properties of the SITAR architecture, an intrusion-tolerant system. Their model does not explicitly represent the attacker or the vulnerabilities that may lead to intrusions, but represents the state of the system in terms of high-level events that may lead to failures. Sheyner et al. [11] have tried to build attack trees automatically using formal methods, and then analyze those trees using Bayesian networks. Ortalo et al. [12] have proposed modeling of known system vulnerabilities using "privilege graphs," followed by a combination of the privilege graphs with simple assumptions about attacker behavior to obtain "attack-state graphs." The latter can be analyzed using Markov techniques to obtain probabilistic measures of security. Singh et al. [13] have used probabilistic modeling to validate an intrusion-tolerant system, emphasizing the effects of intrusions on the system behavior and the ability of the intrusion-tolerant mechanisms to handle those effects, while using very simple assumptions about the discovery and exploitation of vulnerabilities by the attackers to achieve those intrusions. Gupta et al. [14] have used a similar approach to evaluate the security and performance of several intrusion-tolerant server architectures. In most of the above efforts, the designers chose modeling assumptions and model parameterswithout incorporating the justifications for those choices into a larger validation framework. Moreover, the representation of the attacker's behavior is highly simplified. The chief value of existing research in this area is that it demonstrated the applicability of these approaches as evaluation techniques that can be used as components of an integrated assurance argument.

The safety-critical systems community has been using "safety cases" [15] as a means to express arguments about the guaranteed safety of systems such as nuclear power plants. While safety cases allow disparate kinds of evidence to be incorporated into an overall argument, they primarily serve as a visual aid, and do not give formal guidelines as to how an argument/requirement is decomposed, especially if it is quantitative in nature. The models used for generating actual evidence in argument trees are usually simplistic; there is a lack of traceability between the actual design and the models; and quantitative evidence (typically generated by probabilistic models) is generally in the form of a leaf, with no argument subtree enumerating and justifying the assumptions made in the models. Furthermore, an approach based on an integrated argument has not been applied to validation of security-related properties.

We have recently developed and applied a validation method for survivable systems that includes the use of logical arguments, stochastic simulations, and experiment-based testing with actual system components (e.g., red teaming [7]). The need for such diversity is due, in part, to the basic nature of a survivability requirement, which can generally be decomposed into sub-requirements that differ with regard to the types of techniques that can be applied. We applied the method to a proposed intrusion-tolerant design for a publish-subscribe system (hereafter referred to as IT Pub-Sub), both to demonstrate that the proposed design satisfies imposed survivability requirements and to help us consider different protection trade-offs as we developed the final design. The resulting assurance argument incorporates each piece of assurance evidence as a supporting argument of some higher-level claim.

A major component of our validation approach was probabilistic modeling. The probabilistic models used an innovative attacker model. The attacker model has a sophisticated and detailed representation of various kinds of effects of intrusions on the behavior of system components (such as a variety of failure modes). It includes a representation of the process of discovery of vulnerabilities (both in the operating system(s) and in the specific applications being used by the system) and their subsequent exploitation, and considers an aggressive spread of attacks through the system by taking into account the connectivity of the components of the system, at both the infrastructure and the logical levels. We believe that this attacker model is applicable to a wide range of secure and intrusion-tolerant systems. Moreover, we demonstrate the use of probabilistic modeling, as embedded in the IVP, to compare different design configurations, allowing the designers of the system to make choices that maximize the survivability of the system before it is actually implemented.

The remainder of the paper is organized as follows. Section 2 provides an overview of the publish-subscribe system studied. Section 3 provides the outline of our validation procedure. Section 4 provides the details and results of the validation procedure. Finally, we conclude in Section 5 with a summary of lessons learned and prospects for further evolution of our work.

2 OVERVIEW OF THE IT PUB-SUB DESIGN

The IT Pub-Sub system consists of multiple clients communicating with each other through a central core, as shown in Fig. 1. The core consists of three *zones* (or layers) of components: the *crumple zone* (outermost layer), the *operations zone*, and the *ex-*

ecutive zone (innermost layer). The network connectivity is constrained, through the use of network-interface-level hardware firewalls and configurable network switches, to limit direct network connectivity from the outside network to the inner zones. Communication between machines in each zone is accomplished via proprietary communication protocols.

The primary component in the crumple zone is the access proxy (AP). It functions as a bridge between the outer (public) network containing the clients and the inner core network for the remaining zones. It translates between multiple publicly supported communication interfaces (such as RMI and CORBA) and the single internal core communications protocol. Clients connect to the access proxy to publish, subscribe, and query IO. Attacks on clients generate alerts that are forwarded to the core via the AP, and commands to the client security components from the core pass back to the client via the AP.

Inside the crumple zone is the operations zone. It contains the components that perform the two main functions of the core: processing IO objects, and monitoring and maintaining the security of the core and the clients connected to it. IO processing is performed by three operations zone components: the *PSQ server* (PSQ), the *downstream controller* (DC), and the *guardian* (Gu). The PSQ server receives IO objects sent to the core via the AP in the crumple zone. The DC verifies the signatures on messages sent from clients to ensure data integrity. The guardian uses scenario-specific and domain-specific knowledge to identify corruption within the contents of the IO.

Security monitoring and maintenance are performed by local components distributed across all of the hosts in the client, the crumple zone, the operations zone, and a centralized *correlator* (Co) in the operations zone. The components local to individual hosts are *sensors*, *actuators*, and *local controllers* (LC). Sensors are dedicated to intrusion detection, actuators are mechanisms that carry out actions when commanded, and an LC is the control agent responsible for local survivability management.

The correlator receives alerts from multiple intrusion detection sensors within the client, crumple, and operations zones, filters out redundant and false alerts, and forwards serious messages to the system manager. The correlator interprets the alerts it receives in the context of the global state of the system, allowing it to better identify redundant alerts and false alarms.

The executive zone contains the *system manager* (SM). It serves as the master controller of the IT Pub-Sub. It monitors intrusion alerts received from the correlator and generates commands for the appropriate response to counter the intrusion. Human operators monitor the IT Pub-Sub via displays generated by the SM, and can manually initiate specific responses from the SM.

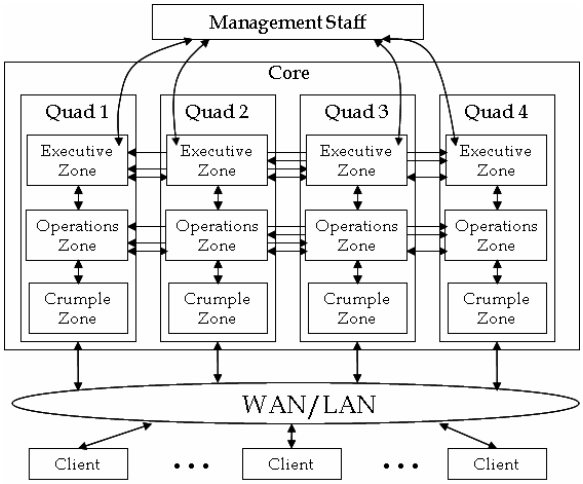


Fig. 1 – The IT Pub-Sub Architecture

The core is redundant, consisting of four *quadrants* (or *quads*) that run different operating systems. Each quadrant contains copies of all crumple, operation, and executive zone components discussed previously. Agreement protocols run among the SM and PSQ servers to ensure a common, collective view of the system state by human operators viewing it through the SM displays, and by clients interacting with the core via the access proxies in the crumple zone. The baseline configuration of the system used in our case study assigned the same operating system to all the hosts within a single quadrant of the core.

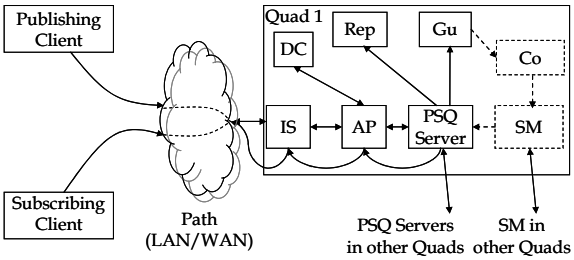


Fig. 2 – Publish Data Flow

Among the various data flows in the system, two are central to the validation effort described in the sections that follow. The first, which is depicted in Fig. 2, is the data flow among the system components during the publish operation. The publish operation begins when a client creates an IO to be published. The IO is signed using the client's private session key and sent to the access proxy in one of the quadrants through the "publish" protocol. The access proxy receives the IO through the isolation switch (IS) and sends it to the DC to verify

whether the client is in a valid session. After successful verification, the AP sends the IO to the PSQ server in its quadrant. The PSQ server forwards the IO to the other PSQ servers in the other quadrants. Each PSQ server then stores the IO in its repository (Rep), sends to the client an acknowledgment of the receipt of the publication, and sends the IO to the guardian. The guardian performs domain-specific tests on the IO. If it finds an error in the IO, it sends an alert to the correlator. The correlator determines, based on the threat level of the alert and the alert state of the system, whether the alert is likely to represent an attack. If it is, the alert is forwarded to the SM. The SMs collectively decide whether they should tell the PSQ servers to recall the IO. If there are no recalls from the SM, the IO is accepted, and subscribing clients are notified that a new IO is available. The subscribing clients then query the system for the available IO.

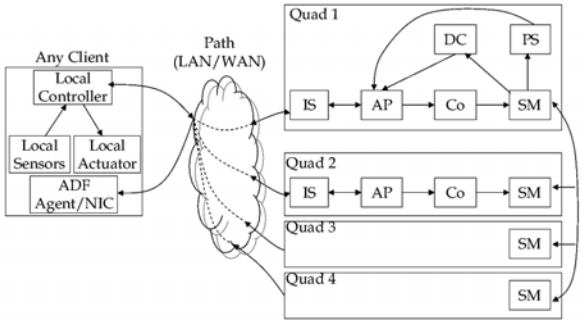


Fig. 3 – Alert/Response Data Flow of a Client

The second data flow of interest is the alert/response data flow associated with the intrusion detection and tolerance capabilities of the system. Alerts are generated on the clients and on components in the core, and the data flow is similar for both cases. The data flow for client alerts is presented in Fig. 3 and described here. When a sensor in a client host detects an anomalous condition/attack, it generates an alert and sends it to the LC on the same host. The LC can either make a local response, such as restarting a process or replacing a corrupted file, or forward the alert to the core for a coordinated adaptive response by the correlator. The correlator then decides (using correlation with other alerts) if this alert is critical, determines the compromised client(s), and informs the SM in its quadrant. The SMs propagate the reports within the SM group and reach a consensus to take action. Each SM then generates a command for the desired response on the client generating the alert and sends it to the client's LC via the DC and AP. The LC commands the appropriate local actuator on the client to take action. If the SM group determines that a more drastic response is appropriate, the client can be quarantined. Each SM notifies the policy server (PS) for the hardware-based firewall system and isolates the compromised client.

3 VALIDATION APPROACH

Before we delve into the the details of the validation, we clarify the nature of high-level survivability requirements and formulate a theoretical basis for their decomposition into sub-requirements. We then provide an outline of the procedure we employed to validate the IT Pub-Sub with respect to particular high-level requirements.

3.1 Survivability Requirements

As defined in [1], *survivability* is the capability of a system to fulfill its mission, in a timely manner, in the presence of attacks, failures, or accidents. A system is *survivable* if it has the above-stated capability according to a specified set of *survivability requirements*. The latter are statements that collectively imply what is meant by the system's "capability to fulfill its mission in an adverse operational environment." (In what follows, it is assumed that "in a timely manner" is accounted for as part of "mission fulfillment.") A survivable system's *operational environment* includes both mission-specific interactions on the part of intended users (its *use environment*) and adverse interactions due to attacks and faults (the *attack/fault environment*).
The term "capability" is particularly important in defining survivability requirements. In some instances, it suffices to identify capability with "certainty," i.e., fulfillment occurs with probability 1. However, capability often needs to be otherwise quantified, and that often involves probabilities. To illustrate this, let us suppose X is some service that is essential to fulfilling the system's mission. Then a corresponding survivability requirement for the system, call it *ExR* for "Example Requirement," could be the following.

ExR: Whenever requested, X is successfully delivered with a probability of at least p_x ($0 < p_x < 1.0$),

where "successfully delivered" needs to be further elaborated in terms relating to both the system and the mission objectives. For example, suppose that X involves the transfer of a data object from one system user to another. Then, successful delivery (provision) of this service, among other things, can depend on the end-to-end data flow required to realize X, timeliness of the data flow, and integrity of the received data object, along with other security properties such as authentication (the sender is an authorized user) and confidentiality (the data object is not disclosed to an unauthorized recipient).

Generally, a system requirement can be viewed as a predicate $R(s)$, where the variable s refers to a system along with relevant aspects of its operational environment. When applied to a specific system S, the proposition $R = R(\mathsf{S})$ is then a *requirement for* S, where *R is satisfied by* S (alternatively, S is *valid with respect to R*) if the truth value of R is **true**. When a requirement R for a particular system is being stated, the system is usually understood from context, and hence not referred to explicitly in the statement of R. (See the definition of *ExR* above, in which we omitted saying "by the system" after "successfully delivered.") This practice will be followed throughout the paper.

With a slight abuse of terminology, we regard a requirement as being *quantitative* if deciding whether R is satisfied by the system entails the evaluation of at least one quantitative measure. In the case of high-level survivability requirements, such quantification is likely probabilistic, as illustrated in the example given above. This is due mainly to uncertainties in the system's operational environment (i.e., use environment and attack/fault environment) together with defense mechanisms that may behave probabilistically. Evaluation of the corresponding measure(s) can be based on a model of the system and its operational environment, experimentation with an actual system (operating in an actual or simulated environment), or some combination of the two. In addition to measure evaluation, other techniques need to be brought into play to determine whether a quantitative requirement is satisfied. If a requirement is *non-quantitative*, then its satisfaction can generally be decided without direct invocation of evaluation results for any quantitative measure. (Indirectly, such results can provide evidence that the requirement is indeed non-quantitative.)

Generally, a requirement is specified as a constraint (typically a bound or equality) on the probabilities of one or more events. This includes non-quantitative requirements, which can be viewed as events with probability 1. Moreover, such probabilities may be *conditional*, meaning that they are conditioned on the satisfaction of one or more preconditions. For example, the validation effort might initially focus on the survivability of the system, assuming it has been successfully bootstrapped. That would make it possible to deal with the bootstrapping process separately, if needed, perhaps later on in the validation procedure. In the case of *ExR*, we may define:

E_{ExR} = X is delivered successfully upon request.

C_{ExR} = The system has been successfully bootstrapped, which includes starting the application that provides the service X.

$ExR = P[E_{ExR}|C_{ExR}] \geq p_X$.

Each event (including preconditions) may be stated as a conjunction of simpler events, both for clarity and ease of subsequent requirement decomposition. Since events are technically sets in the context of probability theory, throughout the paper, "conjunction between events" describes the corresponding set intersection.

3.2 Requirement Decomposition

If feasible, it is helpful to logically decompose a requirement into (more specific) sub-requirements, such that if all of the sub-requirements are satisfied by the system, then the original requirement is satisfied. The primary purpose of the decomposition is to obtain sub-requirements that can be proved using the tools available to the validators for logical argumentation and probabilistic modeling. Our formalism is somewhat similar to the formal composition and refinement framework developed by Abadi-Lamport [16, 17] and Shankar [18]. However, their logical formulations do not have probabilistic underpinnings capable of dealing with quantitative survivability requirements, such as our formalism aims to provide. More precisely, a *logical decomposition* (LD) of a requirement R is a set of two or more requirements (the *sub-requirements of R*) such that their (logical) conjunction implies R. Formally, let $\{R_1, R_2, \ldots, R_m\}$ denote the set of sub-requirements of requirement R ($m \geq 2$) and let $\wedge$ denote *conjunction* (the logic operator AND). Then,

$$(R_1 \wedge R_2 \wedge \ldots \wedge R_m) \Rightarrow R,$$

i.e., the conjunction of the sub-requirements (logically) implies R. Accordingly, in order to validate the system with respect to R, it suffices to validate it with respect to each of the sub-requirements. We rule out trivial conjunctions such as $R \wedge R$ and $R \wedge$ *Taut*, where *Taut* is a tautology (is always **true**). We also rule out degenerate LDs for which the conjunction $(R_1 \wedge R_2 \wedge \ldots \wedge R_m)$ is a contradiction (is never **true**), e.g., the sub-requirements are inconsistent, since in that case (by the definition of logical implication), requirement R would be trivially satisfied.

A requirement is *decomposable* if it admits to an LD. An LD of R is typically determined by finding some initial LD of R and then iteratively determining LDs of decomposable sub-requirements. Note that conditions of the LD definition are preserved by such iterations by the transitivity of $\Rightarrow$.

A requirement R is *atomic* if it is not logically decomposed. This applies to the original requirement (if it is not decomposed) or to any sub-requirement that is not further decomposed. It is important to note that the LD process is guided by the need to

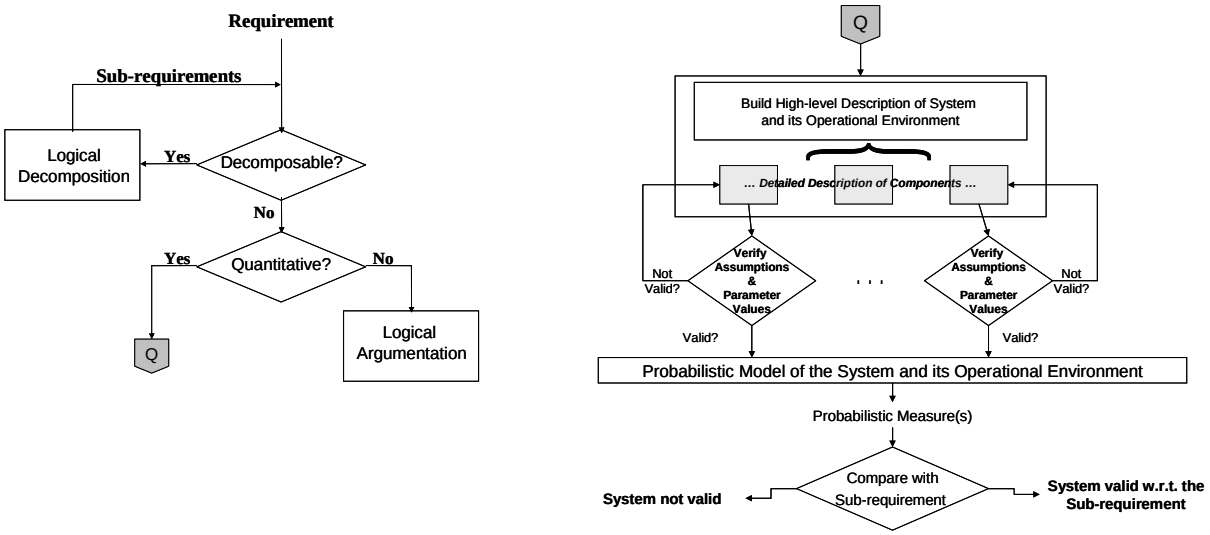


Fig. 4 – A Flowchart Depicting the IVP

obtain manageable sub-requirements, so while a requirement may be atomic because it is not decomposable (does not admit to an LD), in practice, a requirement is often atomic because a beneficial LD is not evident or because the requirement is basic enough to be dealt with effectively without further decomposition. If R is decomposed in the iterative manner described above, the decomposition may be visualized as a tree, with R as the root node and the atomic sub-requirements as the leaf nodes. It follows (due to the preservation of the LD condition) that R is satisfied if all of its atomic sub-requirements (leaf nodes for the tree rooted at R) are satisfied.

Generally, the sub-requirements of an LD of R can be either quantitative or non-quantitative (see Section 3.1). If R is quantitative, then at least one of its sub-requirements must also be quantitative. The reason is that logical decomposition cannot eliminate the quantitative aspect(s) of R. Moreover, it is possible for an LD of a quantitative requirement to contain more than one quantitative sub-requirement. Consider the following example. Suppose we have a requirement R, defined as

$$R\text{: } P[E_1 \wedge E_2] \geq p \ (0 < p < 1),$$

where E_1 and E_2 are two events, such that neither of them is expected to occur with probability 1. Suppose further that these events are not statistically independent. Nevertheless, R can still be logically decomposed into two sub-requirements

$$R_1\text{: } P[E_1] \geq p_1 \ (0 < p_1 < 1),$$
$$R_2\text{: } P[E_2|E_1] \geq \frac{p}{p_1},$$

where it follows from elementary probability theory that $R_1 \wedge R_2 \Rightarrow R$. The value of p_1 can be specified by using a probabilistic model of the relevant portion of the system to evaluate $P[E_1]$ and letting $p_1 = P[E_1]$, thereby satisfying R_1. Another model can then be used to evaluate $P[E_2|E_1]$ so as to determine whether R_2 is satisfied. Note that (due to the use of *implication*), it is sufficient to do this for one value of p_1, and that the independence of the events involved is not a prerequisite for decomposition.

3.3 Outline of the Validation Procedure

Although it is sometimes possible to obtain a validation result by applying a single validation technique (e.g, logical argumentation, if the requirement is not quantitative), we find that validation with respect to high-level quantitative survivability requirements calls for an integrated application of several techniques. Indeed, we believe that the means of accomplishing this is a distinguishing feature of the effort reported herein. We used the following *integrated validation procedure* (IVP) to validate the IT Pub-Sub with respect to a quantitative survivability requirement, say R. The quantified aspects of R are assumed to be probabilistic (e.g., probabilities, moments of random variables, and so forth). The IVP is summarized in Fig. 4, and the steps are:

1. Formulate a precise statement of R, including any assumed preconditions regarding the system and/or its operational environment. The purpose of this step is to make the goals of the validation exercise absolutely clear right at the onset, and also to specify exactly the scope of the validation by explicitly enumerating the preconditions. We chose propositional logic in combination with a simple probabilistic formulation as the specification formalism, since it easily supports subsequent decomposition as described in Section 3.2.
2. If R admits to logical decomposition, decompose it iteratively using the method described in

Section 3.2, thereby determining its corresponding atomic sub-requirements. Otherwise, R is the only atomic requirement. This step is intended to break R into manageable sub-requirements that can be addressed by the tools and techniques used for probabilistic modeling or logical argumentation. Each decomposition of a requirement into sub-requirements is accompanied by a proof of the validity of the decomposition. The proofs would typically use propositional logic, and might use some probabilistic reasoning if any of the involved sub-requirements are quantitative.

The following steps are applied to each atomic sub-requirement R_a. ($R_a = R$ if R is atomic.)

3. If R_a is quantitative, proceed as follows; otherwise, jump to Step 8. In a natural language (or some more formal language suited to the task), describe properties of the system and its operational environment that can guide model-based evaluation of the probabilistic measure(s) associated with R_a. First, the system components (and communications among them) relevant to R_a are identified. This is followed by a description of the following. Note that all the descriptions are in terms of the components and communications identified, i.e., they are specified at the same level of abstraction.
   a) Information flows:
      i) Service-related data flows: this is a block diagram, with the components identified above as the blocks, representing the precise sequence, and the nature, of communications between the components during *normal* operation.
      ii) Attack-caused intrusion detection and recovery flows: this is a block diagram representing the precise sequence, and nature, of communications involved in passing alerts (upon detection of intrusions) and subsequent recovery commands.
      iii) Fault-caused error detection and recovery flows: similar to (ii) above, except that the alerts are raised upon detection of errors.
   b) Use scenario(s): this is closely related to information flows. Here, the operational setting of the system is described, and might include details such as frequencies of various events. A description of the quantitative measure(s) required to evaluate R_a is also provided.
   c) Attack and fault effects: This is a description of the possible effects attacks and faults have on the behavior of the system components. The emphasis is on enumerating the effects that can be detrimental to the system's ability to satisfy R_a. It usually includes a representation of the attacker behavior, in particular the dynamics of the spread of intrusions deeper into the system using already intruded components as launching pads. These descriptions can be based on information from threat and vulnerability assessment and from whiteboarding.
4. Determine detailed descriptions of submodels corresponding to the relevant components identified in Step 3, along with explicit statements of underlying assumptions made for each submodel. For each submodel, descriptions elaborate on (1) the participation of the component in the various information flows described in 3a, (2) the state to be maintained to evaluate the measure described in 3b, and (3) effects of intrusions on the behavior of the component as per 3c. The descriptions are at a level of detail that permits relatively straightforward construction of corresponding portions of the probabilistic model in Step 6. Step 4 also includes identification of the input parameters required by the model, and choice of reasonable estimates for their values.

Steps 3 and 4 help the validators make sure they clearly understand the system design or implementation being validated, and document that understanding. Since the probabilistic model constructed in Step 6 may not be easily understood by persons lacking background in probability theory and stochastic modeling, the descriptions prepared in the above steps can be reviewed by the designers/implementors of the system to ensure compliance with their views. Similarly, the descriptions help convince the accreditors that the models actually represent the system being validated. The above exercise also greatly eases the job of building the probabilistic model.

5. Verify the modeling assumptions of Step 4 and, where possible, justify values of the model parameter values chosen. Since we were validating a system design (as opposed to an implementation), several of the parameter values chosen were based on informal justification rather than formal proofs or experimentation. The focus in such a case is more on exploration of the design/parameter space and identification of the subspace that ensures compliance with the survivability requirements, thus leading to a more survivable system when the chosen design is actually implemented. The assumptions are also checked for consistency with any preconditions in R_a. Furthermore, if a node in the requirement decomposition tree used independence of the underlying events to justify the decomposition, the sets of assumptions in the subtrees rooted at the children of that node are checked against

each other for the violation of the independence assumption.

6. Based on the descriptions obtained in Step 4, construct a probabilistic model of the system and its operational environment that can support evaluation of the probability measure(s) associated with R_a. Several modeling formalisms may be used. We have used Stochastic Activity Networks (SANs) [19], a generalization of stochastic Petri nets, as the modeling formalism. The models were built using the Möbius tool, which can either solve them analytically by converting them into equivalent continuous time Markov chains, or simulate them by executing multiple behavioral trajectories until the measures being evaluated are determined within desired bounds of accuracy.
7. Based on the model made in Step 6, evaluate the probability measure(s) associated with R_a. If the values obtained are within bounds prescribed by R_a, then R_a is satisfied by the system (the system is valid with respect to R_a).
8. If R_a is not quantitative, prove that it is satisfied using logical argumentation.

Note that Steps 4-5 will usually be iterated. For example, an inability to verify some assumption in Step 5 may lead to alternative assumption details in Step 4 (even though realities of the design and its operational environment remain unchanged).

4 VALIDATION DETAILS

As mentioned in Section 2, the essential services provided by the core to clients are *publish*, *subscribe*, and *query*. Accordingly, IT Pub-Sub survivability with respect to these services is a dominant concern of the validation process. We now describe the application of the validation procedure outlined above as it was used to validate the system against the survivability requirement for the publish service.

4.1 Step 1: Requirement Specification

The "capability to process a publish request successfully" was chosen as the survivability requirement for the publish service. The first step in the validation procedure is to formulate a precise statement of the requirement. To this end, the terms "capability" and "successfully process a publish request" need to be defined, together with any preconditions regarding the system and its operational environment.

Let C_{PUB} be the conjunction of the events representing the preconditions. In our case, C_{PUB} is the conjunction of the following two events; the first refers to the system and the second to the use environment.

C^1_{PUB} = the publishing client is successfully registered with the IT Pub-Sub core (authentication).

C^2_{PUB} = the publishing client's mission application always passes adequate and accurate information to the client.

These reflect the assumptions about the system's initial state and invariants during operation, which are taken as axioms in the subsequent analysis.

Let E_{PUB} be the desired event, i.e., the successful processing of a request to publish. It is the conjunction of the following events.

E^1_{PUB} = the data flow of the publish operation is correct.

E^2_{PUB} = the time required for the publish operation does not exceed a specified duration t_{max} (timeliness).

E^3_{PUB} = the published IO that becomes available to subscribers has the same essential content as that assembled by the publishing client (integrity).

E^4_{PUB} = the published IO is available only to the other clients via subscribe or query requests (confidentiality).

Let $P[E_{PUB}|C_{PUB}]$ be the probability of event E_{PUB}, given that the preconditions hold, i.e., the quantification of "capability." Let the required capability be denoted by p_{PUB} ($0 < p_{PUB} < 1$), the lower bound on $P[E_{PUB}|C_{PUB}]$, where its specified value is typically a high probability that depends on the nature of the system use scenario.

The survivability requirement for the IT Pub-Sub publish service, denoted by PUB, can then be stated as

$$\text{PUB: } P[E_{PUB}|C_{PUB}] \geq p_{PUB}.$$

PUB is therefore a quantitative requirement (in the sense described in Section 3.1), since deciding whether it is satisfied by the IT Pub-Sub, entails evaluation of the quantitative measure $P[E_{PUB}|C_{PUB}]$. Due to uncertainties in attacks, attack effects (intrusions), intrusion effects, and the operation of various IT Pub-Sub intrusion tolerance mechanisms, validation is trivial in the case $p_{PUB} = 1$, since we know that PUB cannot be satisfied. At the other extreme, if the value $p_{PUB} = 0$ were allowed and so specified, then PUB would likewise be non-quantitative. Again, validation would be trivial, since, in this case, PUB would always be satisfied (the requirement itself is trivial).

4.2 Step 2: Logical Decomposition

PUB can be initially decomposed into the following two sub-requirements.

$$\text{PUB}_1\text{: } P[E_{PUB}|\, C^2_{PUB}\,] \geq p_{PUB}.$$

PUB_2: $P[C^1_{PUB}] = 1$.

To establish that {PUB_1, PUB_2} is an LD of PUB, we need to show that the defining conditions of an LD are satisfied, i.e.,

$$(PUB_1 \wedge PUB_2) \Rightarrow PUB.$$

Suppose the hypothesis holds, i.e., both PUB_1 and PUB_2 are true. Generally, if A and B are two events such that $P[A] = 1$ then

$$\begin{aligned} P[A \wedge B] &= P[A] + P[B] - P[A \vee B] \\ &= P[B] = P[A]P[B] \end{aligned} \quad (1)$$

where $P[A \vee B] = 1$, since the probability of an "or" event is at least the probability of either disjunct, and can be no greater than 1. From the above identity, it follows that any event having probability 1 is (statistically) independent of an arbitrary event. In particular, by PUB_2, C^1_{PUB} is independent of both E_{PUB} and C^2_{PUB}. Hence,

$$\begin{aligned} P[E_{PUB} | C^2_{PUB}] &= \frac{P[E_{PUB} \wedge C^2_{PUB}]}{P[C^2_{PUB}]} = \frac{P[E_{PUB} \wedge C^2_{PUB}]P[C^1_{PUB}]}{P[C^2_{PUB}]P[C^1_{PUB}]} \\ &= \frac{P[E_{PUB} \wedge C^1_{PUB} \wedge C^2_{PUB}]}{P[C^1_{PUB} \wedge C^2_{PUB}]} = P[E_{PUB} | C^1_{PUB} \wedge C^2_{PUB}] \\ &= P[E_{PUB} | C_{PUB}] \end{aligned}$$

Since $P[E_{PUB} | C^2_{PUB}] \geq p_{PUB}$ (sub-requirement PUB_1), by the above identity we can conclude that

$$P[E_{PUB} | C_{PUB}] \geq p_{PUB},$$

which is just the original requirement PUB. Hence, the LD condition holds.

Regarding the four events that define E_{PUB}, a study of the design reveals that integrity (E^3_{PUB}) and confidentiality (E^4_{PUB}) can likewise be regarded as probability-1 events, given that precondition C^2_{PUB} is satisfied (event C^2_{PUB} occurs). More precisely, we have the following sub-requirements of requirement PUB_1.

$PUB_{1,1}$: $P[E^1_{PUB} \wedge E^2_{PUB} | E^3_{PUB} \wedge E^4_{PUB} \wedge C^2_{PUB}] \geq p_{PUB}$
$PUB_{1,2}$: $P[E^3_{PUB} | C^2_{PUB}] = 1$
$PUB_{1,3}$: $P[E^4_{PUB} | C^2_{PUB}] = 1$

To insure that {$PUB_{1,1}$, $PUB_{1,2}$, $PUB_{1,3}$} is an LD of PUB_1, we must again show that the constraint for LD holds, i.e.,

$$(PUB_{1,1} \wedge PUB_{1,2} \wedge PUB_{1,3}) \Rightarrow PUB_1.$$

Suppose the above hypothesis holds ($PUB_{1,1}$, $PUB_{1,2}$, and $PUB_{1,3}$ are true). Extending observation (1) to conditional probabilities, it follows from $PUB_{1,2}$ and $PUB_{1,3}$ that $P[E^3_{PUB} \wedge E^4_{PUB} | C^2_{PUB}] = 1$. Using this fact together with the sub-requirements $PUB_{1,2}$ and $PUB_{1,3}$,

$$\begin{aligned} P[E^1_{PUB} \wedge E^2_{PUB} | E^3_{PUB} \wedge E^4_{PUB} \wedge C^2_{PUB}] &= \frac{P[E^1_{PUB} \wedge E^2_{PUB} \wedge E^3_{PUB} \wedge E^4_{PUB} \wedge C^2_{PUB}]}{P[E^3_{PUB} \wedge E^4_{PUB} \wedge C^2_{PUB}]} \\ &= \frac{P[(E^1_{PUB} \wedge E^2_{PUB} \wedge E^3_{PUB} \wedge E^4_{PUB}) \wedge C^2_{PUB}]}{P[E^3_{PUB} \wedge E^4_{PUB} | C^2_{PUB}]P[C^2_{PUB}]} \\ &= \frac{P[E_{PUB} \wedge C^2_{PUB}]}{1 \cdot P[C^2_{PUB}]} = P[E_{PUB} | C^2_{PUB}] \end{aligned}$$

Since $P[E^1_{PUB} \wedge E^2_{PUB} | E^3_{PUB} \wedge E^4_{PUB} \wedge C^2_{PUB}] \geq p_{PUB}$ by $PUB_{1,1}$, we conclude from the above identity that PUB_1 is true.

Each sub-requirement available at this stage is specific enough to be handled by a validation technique such as probabilistic modeling or logical argumentation. Hence, we stop the decomposition, and refer to $PUB_{1,1}$, $PUB_{1,2}$, $PUB_{1,3}$, and PUB_2 as the atomic sub-requirements of PUB, where the first is quantitative and the other three are non-quantitative.

4.3 Step 3: High-Level System Description

For the remaining steps of the validation, we will focus on the quantitative atomic requirement $PUB_{1,1}$. With respect to $PUB_{1,1}$, this step seeks high-level descriptions of the IT Pub-Sub data flow for the publish operation, the associated alert and response data flows, and the attack/intrusion environment. The descriptions guide subsequent determination of the detailed descriptions needed to construct a probabilistic model that supports evaluation of the probability measure $P[E^1_{PUB} \wedge E^2_{PUB} | E^3_{PUB} \wedge E^4_{PUB} \wedge C^2_{PUB}]$ associated with $PUB_{1,1}$.

4.3.1 Data Flows

Both the publish data flow and the attack/alert data flow are represented as block diagrams depicting both the relevant components and the sequence of operations and nature of the communication between the components. The diagrams (Fig. 2 and Fig. 3) and the associated descriptions have already been provided as a part of Section 2 and are not repeated here.

4.3.2 Attack Model

The attack model makes several important distinctions concerning where attacks occur (location of both the source and target of an attack) and how resulting intrusions affect both system and attacker behavior. In particular, the model accounts for the fact that once a vulnerability has been discovered in

a target, the attack can quickly propagate to other instances of that target, provided that they are accessible (via network connectivity) from the attack source. If an intruded target (e.g., a host) is compromised, then it is possible for the host to serve as a source of further attacks.

Terminology In order to describe the attack model more precisely, we make the following distinctions. An entity of the system is one of the following:

- *A host*: A computing resource with an operating system and network interface cards.
- *A component*: A process that realizes an IT Pub-Sub function, e.g., an access proxy. Note that several components typically reside in a single host (for example, the survivability delegate, the sensors, the actuator, and the local controller reside in the client).
- *A process domain (PD)*: An entity that implements a component, e.g., the sensor process domain implements the sensor component, and the AP IO (AP_IO) process domain implements the AP that deals with forwarding IOs.
- *An application*: The application level of the process domain.

An intrusion is a possible outcome of an attack. It occurs if the attack finds a vulnerability in its target and thereby alters the target's behavior (the effect or symptom of the intrusion); otherwise, it is prevented. In other words, an intrusion occurs if an attack has some effect on the target. The effect can range from something very benign (e.g., when the intrusion is "masked" or "blocked") to the compromise of the target such that it can be used as a platform for launching further attacks. An intrusion is *prevented* if the attack can find no vulnerabilities in its target, thereby obviating any effect. In particular, if an attack is unable to access its target, then the attack cannot find a vulnerability, even if one exists.

An intrusion is *tolerated* (possibly in the presence of other tolerated intrusions) if its effect does not lead to unsuccessful processing of a publish request; otherwise, it causes a failure.

A *new vulnerability*, found at time *t*, is a vulnerability that is present in at least one component of the architecture, and that was not known by the attacker until time *t* during the mission. When discovered, such a vulnerability can be used any number of times (until the end of the mission) against the vulnerable components that the attacker can reach.

A successful attack is *repeated* if that attack (same source, same type) is made on a similar target having the same vulnerability, in which case it succeeds very quickly.

A successful attack is *propagated* if the intruded target is compromised such that the target becomes a source of further attacks. If this source can access a similar target with the same vulnerability, the original attack can be repeated (see above). The source can also launch a new attack that attempts to find a new vulnerability in another target.

The simulation model considers attacks that are "successful" in the sense that an intrusion occurs and, moreover, is neither masked nor blocked. However, if such an intrusion is tolerated (the third case noted above), the attack does not succeed in the more usual sense of causing a failure.

Attack Propagation Two basic assumptions underlie the construction of the attack model. First, we assumed that the attacker would discover new vulnerabilities slowly. Define *MTTD* to be the mean time to discovery of a new vulnerability. Second, it was assumed that the attacker would exploit newly discovered vulnerabilities quickly. Once an entity is intruded following the discovery of a vulnerability, the attack can be repeated (see the preceding terminology). Define *MTTE* to be the mean time between successive exploitations of a known vulnerability. The typical value of *MTTE* in our analysis was 5 minutes. It is important to note, however, that repeated attacks require targets with the same vulnerability, typically entities that are instances of the original target. Accordingly, design diversity can be used to reduce the possibility of repeated attacks. For example, a successful OS-level attack from the outside, compromising a client running under OS1, can propagate to hosts connected to the client (namely, the access proxy in the core). If the access proxy's OS is also OS1, the attack can be repeated, with success (intrusion) coming quickly. On the other hand, if the access proxy is running under a different operating system, then the OS diversity will likely preclude a repeated attack. It is possible that a given vulnerability exists in more than one OS. We account for that possibility using a probability of a common-mode vulnerability. If a vulnerability is determined to be common-mode, it will exist in all OSes used by the IT Pub-Sub.

Types of Attacks Three different types of attacks were represented in the attack model.

- *Infrastructure-level attacks* exploit a vulnerability found either in the operating system running on a given host (for instance, a flaw in the TCP/IP stack), or in a service running on that host, not related to the IT Pub-Sub (for example, a flaw in sshd). The attack's source and target must be directly connected and communicate with each other, typically with standard network protocols. If a vulnerability of that type is discovered, it can be limited to one of the four operating systems, or affect all operating systems used in the system.
- *Data-level attacks* exploit a vulnerability in an IT Pub-Sub application on the targeted compo-

nent. The attack's source and target are applications on different hosts. The attack uses the content of the application data to intrude into the target. For example, the client sends a corrupted IO to a PSQ server, resulting in a crash or a corruption of that host.

- *Attacks across process domains* allow the attacker to intrude into a different process domain of the same host by exploiting low-level vulnerabilities in the operating system. The source and targets are process domains running on the same machine.

Successful infrastructure-level attacks are attacks in depth; the intruder quickly progresses deeply into one quadrant, because only one vulnerability needs to be discovered to compromise the common operating system in the entire quadrant. However, unless the vulnerability found corresponds to a common mode failure, the attacker cannot intrude into any other quadrant, because the other quadrants are based on hosts with different operating systems. Data-level attacks provide attacks in breadth; if a vulnerability is found in the access proxy application, then the publishing client can exploit it directly on the four access proxies. Those attacks are consequently much more dangerous. Great effort should be put into reducing the number of data vulnerabilities, and preventing attackers from exploiting them.

The first attack on the IT Pub-Sub must be an infrastructure-level attack, since both of the other types require control of a machine within the system. For example, attacks across process domains assume a compromised process domain for the source of the attack. Data-level attacks require control of a component that is capable of generating IT Pub-Sub protocol packets, signed with signatures from host-specific private keys.

Fig. 5 provides an example of attack propagation. At time t_0 = 85 minutes, an infrastructure-level vulnerability (ILV) is found on the main process domain of OS1. After a short time, the attacker exploits that vulnerability on the publishing client (time t_1). From the client, he launches the same attack on the AP of quadrant 1 at time t_2. He continues in the same way until he has compromised all the components of quadrant 1 (since they're all running the same OS). At time t_8 = 230 minutes, a data-level vulnerability (DLV) on the PSQ server is found. The attacker uses that vulnerability to compromise the three remaining PSQ servers; he can attack either from the publishing client (times t_9 and t_{11}), or from the PSQ server in the first quadrant (time t_{10}), since he has control of all those entities.

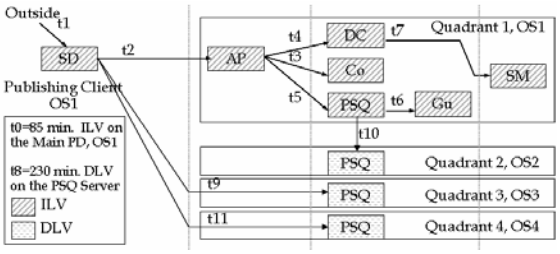


Fig. 5 – Example of Attack Propagation

Intrusion Effects When any of the three types of attacks described above is successfully exploited, the resulting intrusion can have one of the following effects.

- *Crash of the intruded entity*: For the operating systems that handle process domains (*secure OSes*), a crash of a given process domain will not lead to the crash of other process domains. On a *traditional OS* (without process domains), a crash of a given process will crash the whole machine (and thus all the processes running on it).
- *Compromise of the intruded entity*: The intruded entity can corrupt or drop data packets, or can be a source of further attacks. When a process domain becomes compromised on a secure operating system, the other process domains are unaffected. However, for traditional operating systems, compromise of one process also compromises the other processes on the machine.
- *Denial of service*: The intruded entity can reduce processing speeds and increase latencies.

Attack Responses The model includes three mechanisms for responding to a detected intrusion that is not blocked:

- *Type 1: Rapid response*. The first and fastest mechanism is a local and rapid response, based on an automatic decision made by the LC. Examples of rapid response include file restoration, termination of illegal processes, and restarting of critical processes. On average, this operation takes one minute.
- *Type 2: Secure reboot*. This operation is carried out in two phases: automatically shutting down the host and then manually restarting it. The shutdown is performed by the SM through DC and LC. Restarting has to be done manually by the operator sitting at the console. The reason is that the client application will also need to be restarted, which will require the user to perform some action, such as providing a password or swiping a smart card. Moreover, the shutdown will give the client operator the opportunity to introduce diversity, perhaps by removing some services or changing the firewall policies.

- *Type 3: Permanent isolation.* The decision to isolate is taken by the SM through the PS. The decision is made by the SM in the core; therefore, this operation requires more time, on average 7 minutes.

4.4 Step 4: Detailed Descriptions

This step elaborates the Step 3 descriptions to an extent that permits relatively straightforward construction of corresponding components of the probabilistic model. It also explicitly enumerates all the assumptions the probabilistic model makes that are verified, and, if found to be correct, justified, using logical arguments. For the purpose of illustration, we will limit our attention to the AP component, along with certain details concerning the use and attack models.

If the AP is not compromised, it forwards designated traffic from the Quadrant Isolation Switch to core quadrant components and vice versa. The AP, like several other components in IT Pub-Sub, has *process domains*, which are similar to sandboxed virtual machines running on the same host. The AP handles different types of traffic, and each type, including IOs sent from publishing clients to the core, is handled by a different process domain. If an IO is sent by a publishing client to the AP (the client sends the IO to a randomly chosen quadrant), the token accompanying the IO is sent to the quadrant's DC to assist in the determination of whether the publishing client is in a session. If the DC's response is positive, the IO is forwarded to the PSQ server.

In addition, the AP forwards sensor alerts from client-side sensors to the correlator, heartbeats from client components to the DC, acknowledgments from PSQ servers to clients, notifications from PSQ servers to subscribing clients, commands from the DC to clients' LCs, and commands from ADF policy servers to clients' ADF NICs.

The following assumptions are identified in the model for the AP component:

AP1: Only well-formed traffic is forwarded by a correct AP.

AP2: An AP can change the traffic through it, but cannot re-sign the content.

AP3: An AP cannot access the contents of an IO if application-level end-to-end encryption is being used.

AP4: If the AP is compromised, it can launch ILA to the following components: client, PSQ, DC, correlator, and PS.

AP5: If the AP is compromised, it can launch DLA to the following components: PSQ, guardian, DC, and the publish-subscribe middleware components, as well as the IDS components on the clients, correlator, and SM.

The *time-to-discovery* (TTD) and *time-to-exploit* (TTE) are assumed to be exponentially distributed. Accordingly, the values assigned to *MTTD* and *MTTE* (see the description of the attack model above for definition) parameters of the attack model will be such that *MTTE* « *MTTD*. This reflects the fact that the TTD of a new vulnerability is typically much longer than its subsequent TTE.

Once an intrusion has occurred, it has one of the three possible effects described in Step 3 (crash, compromise, or DoS), which we assume will occur with probabilities (parameters of the attack model) p_{cr}, p_{co}, and p_{dos}, respectively. When a process domain on an AP is compromised, data that flows through it can be altered as follows. If any alert and command traffic passes through, it is dropped (blocked). If the process domain handles IOs, it will corrupt an IO if it has access to the client's key (since the process domains can change the signature of the IO according to the corrupted content); otherwise, it will drop the IO. It can also act as a source for further attacks (both ILA and DLA) on other entities. Compromised sensors are not able to detect intrusions. When an alert is generated by the IDS, the recovery options are as follows. On the first alert, the affected process domain is restarted. On the next three, the entire AP is rebooted. On the fifth alert, the AP is quarantined (using the ADF NICs).

4.5 Step 5: Justification of the Assumptions

The goal in this step is to justify the model assumptions made in Step 4. We used logical argumentation in this step. Other techniques, such as formal methods and experimentation, were also utilized for some of the assumptions. In the complete description of the model in Step 4, a total of 31 assumptions were justified using logical argument and experimental results. These arguments use very low-level system details, and, in the interest of space, we do not list them here. Interested readers can refer to [20] for details.

4.6 Step 6: Construction of the Probabilistic Model

Based on detailed descriptions of the type illustrated in Step 4, a probabilistic model was constructed using Möbius [21] that supports evaluation of the probability measure $P[\, E^1_{PUB} \wedge E^2_{PUB} \,|\, E^3_{PUB} \wedge E^4_{PUB} \wedge C^2_{PUB}\,]$. We used *stochastic activity networks* (SANs) [19] as the formalism. Behaviorally, a model constructed with SANs represents (meas-

ure-relevant) behavior of the IT Pub-Sub platform in the presence of publish demands and random attack-caused intrusions. Structurally, the *atomic* SAN submodels represent various components of the design and its use/attack environment; descriptions of them were documented during Step 4. The overall model is constructed by using *replicate* and *join* operations to compose the atomic submodels.

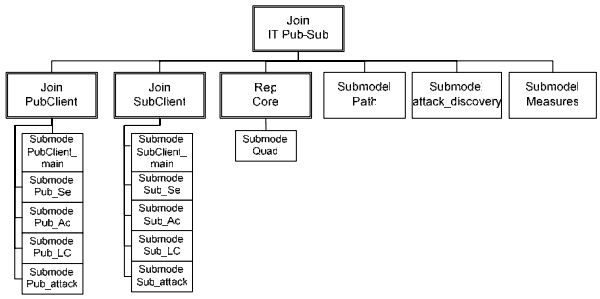


Fig. 6 – Composed Model of the IT Pub-Sub

Graphically, a composed model can be viewed as a tree, in which the atomic SAN submodels correspond to leaf vertices of the tree, and joins or replicates correspond to internal vertices. A *Join* vertex combines two or more different SAN submodels, each of which can itself be a composed model (represented by a subtree). A *Rep* (or Replicate) node generates multiple copies of its submodel (again, each submodel can itself be a composed model). Different submodels in a composed SAN interact through *shared* state variables.

As depicted in Fig. 6, the system is viewed (for the purpose of validating the requirement $PUB_{1,1}$) as two clients (Join *PubClient* and *SubClient*) communicating with the core (Rep *Core*) through the network (submodel *Path*). The clients have four *process domains*, represented by four submodels under the Join (IT Pub-Sub publish functionality, the sensors, the actuator, and the local controller). The fifth submodel under each of these Joins is the attack model. The two remaining submodels (*measures* and *attack_discovery*) assist model-based formulation of the measures evaluated.

The core is a replication of four quadrants (submodel *Quad*). As shown in Fig. 7, each quadrant is a Join of several submodels (*Access Proxy*, *PSQ*, *DC*, *Guardian*, *Correlator*, *PS*, and *SM*), representing all of the core components. As presented in Fig. 1, the access proxy, PSQ server, downstream controller, and guardian have IDS components; therefore, their respective Joins have IDS submodels (for instance, *AP_Se*, *AP_Ac*, and *AP_LC*).

For each node represented in the composed models of Fig. 6 and Fig. 7, an atomic SAN model is built. Atomic SANs encode the state variables representing the components they are modeling, and provide transitions with specified delay distributions that can change the state. They provide the ability to include complex enabling functions for the transitions, and complex completion functions to manipulate the state upon completion of transitions. Each atomic SAN basically implements the description for the corresponding component in Step 4. A detailed description of all the atomic SANs constructed is provided in [22].

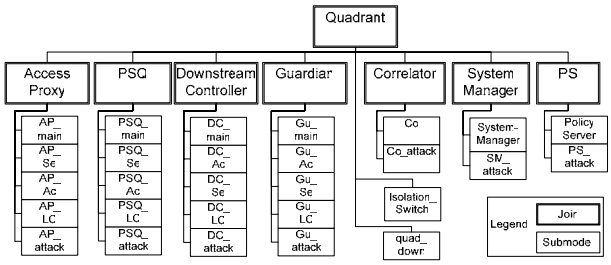


Fig. 7 – Composed Model of the IT Pub-Sub Core Quadrant

The point to note is that the entire model construction is driven by the measure we intend to evaluate. In particular, the SANs have state variables maintaining a count of the number of publish attempts by the clients, and the number of those attempts for which the events $E^1_{PUB} \wedge E^2_{PUB}$ occur. Furthermore, the construction of the model always assumes E^3_{PUB}, E^4_{PUB}, and C_{PUB} are given, and thus does not include any state for keeping track of those events.

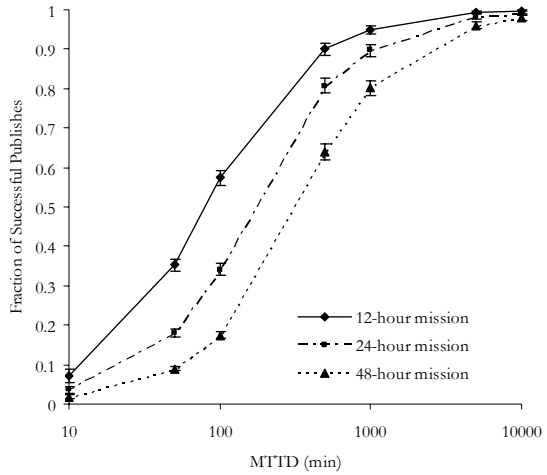
4.7 Step 7: Evaluation for Validation Against Requirement

Based on the model of step 6, this step evaluates the probability measure

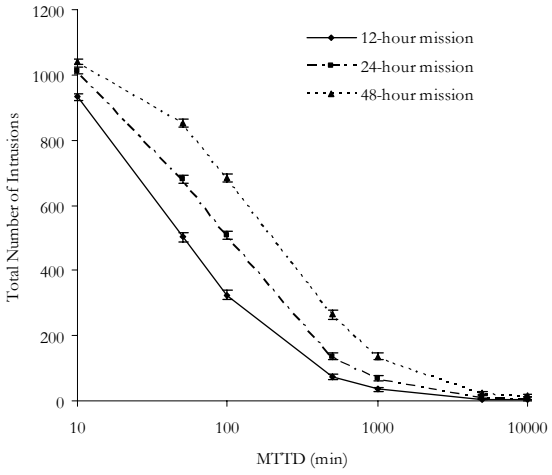
$$p_{1,1} = P[\, E^1_{PUB} \wedge E^2_{PUB} \,|\, E^3_{PUB} \wedge E^4_{PUB} \wedge C^2_{PUB} \,]$$

associated with the atomic quantitative requirement $PUB_{1,1}$. If $p_{1,1} \geq p_{PUB}$ (the required lower bound), then $PUB_{1,1}$ is satisfied by the system (the system is valid with respect to $PUB_{1,1}$).

The IT Pub-Sub system is envisaged to be used for short (of the order of a day or two) durations. Letting d_M denote the duration (in hours) of a use scenario, $p_{1,1}$ is formulated as the fraction of publication requests during d_M that are processed successfully, i.e., the fraction for which the event $E^1_{PUB} \wedge E^2_{PUB}$, given E^3_{PUB}, E^4_{PUB}, and C^2_{PUB}, occurs. For three different choices of d_M, Fig. 8(a) displays the values of $p_{1,1}$ as a function of the *MTTD* of a new vulnerability (in minutes). Those vulnerabilities are assumed to be exploited very quickly, i.e., *MTTE* = 2 minutes. The publishing client publishes a new IO every 5 minutes.



(a) $p_{1,1}$ versus *MTTD* (minutes)



(b) Total number of intrusions versus MTTD (minutes)

Fig. 8 – Variation of Measures with Change in *MTTD*

In particular, if (1) new vulnerabilities are discovered at an *average* rate of no more than once per day (*MTTD* $\geq$ 1440 minutes), which is a fairly aggressive assumption, (2) the value of p_{PUB} used is 0.95, and (3) the mission duration is 12 hours, then the IT Pub-Sub is valid with respect to $PUB_{1,1}$.

In addition to measures associated directly with quantitative survivability requirements, the model constructed in Step 5 is sufficiently detailed to support evaluation of a variety of other survivability-related measures. For example, to understand how the number of successful attacks (those that cause intrusions) relates to a value of $p_{1,1}$, the measure

n = the total number of intrusions during d_M

can likewise be evaluated as a function of *MTTD*.

Comparing Fig. 8(a) with Fig. 8(b), we see how the probability $p_{1,1}$ of successfully processing a publish request during d_M varies inversely with the total number n of successful intrusions experienced during a mission of the same duration.

4.8 Step 8: Satisfaction of Non-quantitative Atomic Requirements

Validation with respect to the non-quantitative atomic requirements $PUB_{1,2}$, $PUB_{1,3}$, and PUB_2 is accomplished using logical argumentation. Given that validity has been established with regard to the quantitative atomic requirement $PUB_{1,1}$ in Step 7 above, the system is then valid with respect to the original requirement PUB. The argumentation used is similar to that used for validation of assumptions, and in the interest of space we do not provide the details here. Details are presented in [20].

4.9 Other Evaluation Results: Exploring Design Choices

In addition to validating the system against the high-level survivability requirements, we also used the probabilistic model developed as a part of the IVP application to study design trade-offs and explore the behavior of the system in different operating configurations and attack environments. Such studies are possible because each component of the system model contains a set of input parameters that define aspects of the component's functionality, such as the rate of data publishes or the type of operating system used on a host. These are in addition to parameters that define the characteristics of the attackers on the system, such as the time between vulnerability discoveries (*MTTD*) and the time to exploit known vulnerabilities (*MTTE*).

We present results for two of the studies performed: effect of the degree of operating system diversity among the four quadrants in the core and effect of the autonomous distributed firewall (ADF NIC) policies.

4.9.1 Effect of Diversity

In this experiment, we studied two different OS configurations in the following way:

- *4 OS:* the access proxy, downstream controller, PSQ servers, and system manager in a given quadrant run the same operating system. Each quadrant uses a different OS. The policy server is available in only one operating system. Therefore, there is no possible variation for this component. This case corresponds to the default configuration of the system.
- *1 OS:* all the components in the core (apart from the policy servers) run the same operating system.

Also, to make a fair comparison between the two options, all OSes are secure.

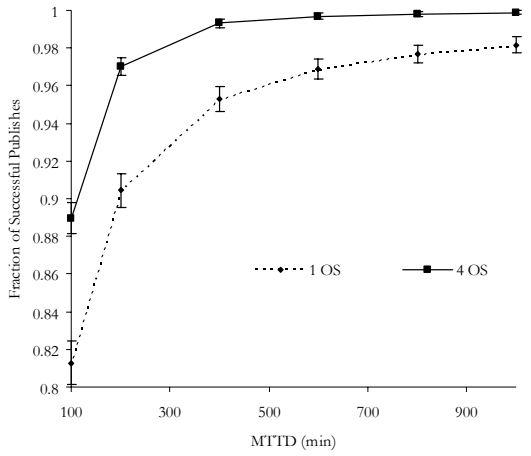


Fig. 9 – $p_{1,1}$ versus MTTD: 4 OS vs 1 OS

The graph presented in Fig. 9 is based on an assumption of a less aggressive attacker than we considered in the baseline case; here, the attacker can only execute infrastructure attacks and attacks across process domains. The figure illustrates the difference between the two options mentioned above. It shows $p_{1,1}$ versus the vulnerability discovery rate. Diversity significantly increases the performance of the design: for *MTTD* = 200 minutes, $p_{1,1}$ is about 0.97 for the 4-OS case, versus 0.90 for 1 OS, i.e., a 70% improvement of the unavailability (0.10 versus 0.03). The gap between the two curves is noticeable at all rates.

A similar experiment was done for all types of attack, including the data-level attacks. In that case, the two curves were closer. Data-level attacks are the most dangerous type of attack, as they can take out the same component in every quad. For example, a compromised client could launch a data-level attack against the four PSQ servers, which could result in the crash (or compromise) of all four. If that happened, no further PSQ requests would be handled, and the core would be considered down. For the results presented here, we assumed that the data-level vulnerabilities could be considerably reduced not only by the effort put into the implementation of, for example, the PSQ, but also by the semantic checks done on the access proxy for any incoming traffic to the core.

4.9.2 ADF NIC Policies

ADF NICs are local firewalls on each component, administered by the policy servers in the core. The third experiment compares three ADF NIC policies, assuming that only infrastructure-level attacks and attacks across process domains are allowed. The first policy is to allow all communications between any two processes of any two components. The second is a per-component policy, allowing only certain components to communicate with each other (for instance, the AP can talk to the PSQ server in its quad, but the client cannot communicate directly with any PSQ server). Finally, the third one is a per-process-domain policy, restricting communications between specific processes.

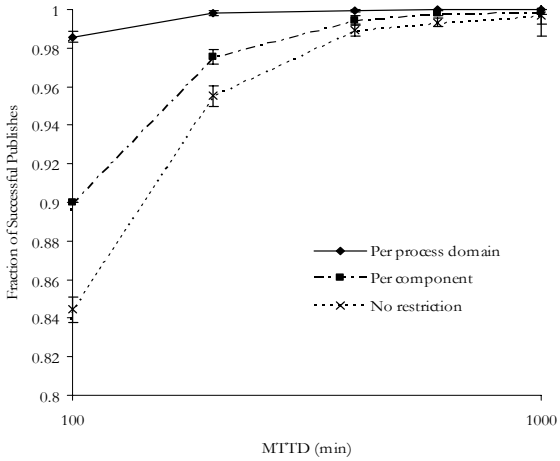


Fig. 10 – $p_{1,1}$ versus MTTD: ADF Policy Study

Fig. 10 presents the results reflecting the three configurations. The graphs reveal that the per-process-domain policy is by far the best of all three: when *MTTD* = 100 minutes, $p_{1,1}$ = 84.4% for the no-restriction policy, versus 90.0% for the per-component, and 98.5% for the per-process-domain one, which corresponds to a 90% improvement of the unavailability (from 15.6% down to 1.5%).

The per-process-domain restriction can also be interpreted as having a ruleset that describes which ports from which machines can communicate with which ports of the other machines. It is a very successful way to increase the survivability of the system, and therefore should be implemented. However, it comes with a price, as it limits the developers by forcing them to allocate fixed port numbers, and also might limit the usability of the machines for purposes other than the PSQ functionalities.

5 CONCLUSION

The complexity of emerging survivable systems, especially those that make use of multiple security approaches and technologies, calls for an integrated approach to survivability validation. We have presented the validation of a survivable publish-subscribe system using a top-down approach that begins with a precise formulation of a specific survivability requirement, and then systematically decomposes the problem into manageable tasks. As

a part of the procedure, stochastic models of the system and of the attacker were presented. We conducted model-based experiments that evaluated the survivability of the system when stressed by various types of attacks by measuring the probability of success for the transactions between the clients and the core. The results show that if the average time between discoveries of new vulnerabilities is longer than one day, more than 95% of the publishes are processed correctly. The system model was used to study design trade-offs, one of which was that OS diversity in the design significantly improved the performance. Another design trade-off became apparent when we compared three ADF NIC policies: a per-process-domain policy leads to the highest availability, but constrains developers.

We described a sophisticated attacker behavior model that has wide applicability when using probabilistic modeling techniques for evaluating large networked information systems.

The integrated validation procedure (IVP) used in this work provided a collaborative environment in which a team of individuals with varied areas of expertise was able to work efficiently and optimally to produce an assurance argument that was convincing to the accreditors of the above effort. The IVP gave the designers of the system several insights into the relative merits of different protection trade-offs by comparing various algorithms, features, or infrastructures. The IVP also brought out hidden assumptions and residual requirements that forced the designers to consider issues that they might otherwise have overlooked. We hope the outlined procedure can be used by other security researchers and practitioners to validate similar survivable systems with respect to high-level quantitative survivability requirements.

The work described in this paper is an instance of an evolving validation methodology. In the future, we plan to use the IVP to validate other intrusion-tolerant systems to ascertain and further refine its generic applicability, using an even wider array of evaluation techniques as the building blocks of the assurance argument. We also intend to explore avenues for further automation, such as in the decomposition of requirements, identification of appropriate assumptions, and presentation of the completed argument.

Acknowledgments

This research was supported by DARPA contract number F30602-02-C-0134. We would like to thank Franklin Webber and the other members of the DPASA team for the development of the system design and their contribution to the attack model. We would like to thank Steve Dawson, Joshua Levy, and Robert A. Riemenschneider at SRI International and Charles Payne at Adventium Labs for their work on the logical arguments. We would like to thank our colleagues at DARPA, particularly Jay Lala and Lee Badger, and at AFRL, particularly Patrick Hurley, for their constructive comments and support of this work. We would also like to thank Jenny Applequist for her editorial assistance.

6 REFERENCES

[1] R. J. Ellison, D. A. Fisher, R. C. Linger, H. F. Lipson, T. Longstaff, and N. R. Mead. Survivable Network Systems: An Emerging Discipline. Technical Report CMU/SEI-97-TR-013, CMU Software Engineering Institute, November 1997.

[2] M. Cukier, J. Lyons, P. Pandey, H. V. Ramasamy, W. H. Sanders, P. Pal, F. Webber, R. Schantz, J. Loyall, R. Watro, M. Atighetchi, and J. Gossett. Intrusion Tolerance Approaches in ITUA. In *Supplement of the 2001 International Conference on Dependable Systems and Networks*, pages B-64-B-65, Göteborg, Sweden, July 2001.

[3] Y. Deswarte, L. Blain, and J. C. Fabre. Intrusion Tolerance in Distributed Computing Systems. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 110-121, May 1991.

[4] B. Dutertre, V. Crettaz, and V. Stavridou. Intrusion-Tolerant Enclaves. In *Proceedings of the IEEE International Symposium on Security and Privacy*, pages 216-224, Oakland, CA, May 2002.

[5] US Department of Defense Trusted Computer System Evaluation Criteria ("Orange Book"). http://www.radium.ncsc.mil/tpep/library/ rainbow/5200.28-STD.html, December 1985. DoD 5200.28-STD.

[6] C. Landwehr. Formal Models for Computer Security. *Computer Surveys*, 13(3):247-278, September 1981.

[7] J. Lowry. An Initial Foray into Understanding Adversary Planning and Courses of Action. In *Proceedings of the DARPA Information Survivability Conference and Exposition II (DISCEX'01)*, pages 123-133, 2001.

[8] B. Littlewood, S. Brocklehurst, N. Fenton, P. Mellor, S. Page, D. Wright, J. Doboson, J. McDermid, and D. Gollmann. Towards Operational Measures of Computer Security. *Journal of Computer Security*, 2(2-3):211-229, 1993.

[9] E. Jonsson and T. Olovsson. A Quantitative Model of the Security Intrusion Process Based on Attacker Behavior. *IEEE Transactions on Software Engineering*, 23(4):235-245, April 1997.

[10] B. B. Madan, K. Goševa-Popstojanova, K. Vaidyanathan, and K. S. Trivedi. Modeling and Quantification of Security Attributes of Software Systems. In *Proceedings of the 2002 International Conference on Dependable Systems and Networks (DSN 2002)*, pages 505-514, June 2002.

[11] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. Wing. Automated Generation and Analysis of Attack Graphs. In *Proceedings of the 2002 IEEE*

Symposium on Security and Privacy, pages 273-284, May 2002.

[12] R. Ortalo, Y. Deswarte, and M. Kaâniche. Experimenting with Quantitative Evaluation Tools for Monitoring Operational Security. *IEEE Transactions on Software Engineering*, 25(5):633-650, 1999.

[13] S. Singh, M. Cukier, and W. H. Sanders. Probabilistic Validation of an Intrusion-Tolerant Replication System. In *Proceedings of the 2003 International Conference on Dependable Systems and Networking (DSN-2003)*, pages 615-624, San Francisco, CA, 2003.

[14] V. Gupta, V. Lam, H. V. Ramasamy, W. H. Sanders, and S. Singh. Dependability and Performance Evaluation of Intrusion-Tolerant Server Architectures. In *Proceedings of LADC 2003: The 1st Latin American Symposium on Dependable Computing*, *Lecture Notes in Computer Science*, volume 2847, pages 81-101, São Paulo, Brazil, October 2003.

[15] P. G. Bishop and R. E. Bloomfield. The SHIP Safety Case Approach. In *Proceedings of the 1995 IFAC Conference on Computer Safety, Reliability and Security (SafeComp95)*, pages 437-451, Belgirate, Italy, October 1995.

[16] M. Abadi and L. Lamport. Conjoining Specifications. Technical Report 118, Digital Equipment Corporation Systems Research Center, December 1993.

[17] L. Lamport. Temporal Logic of Actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872-923, May 1994.

[18] N. Shankar. A Lazy Approach to Compositional Verification. Technical Report TSL-93-08, SRI International, 1993.

[19] J. F. Meyer, A. Movaghar, and W. H. Sanders. Stochastic Activity Networks: Structure, Behavior, and Application. In *Proceedings of the International Conference on Timed Petri Nets*, pages 106-115, Torino, Italy, July 1985.

[20] W. H. Sanders. CDR validation report. Technical Report CDRL A007-R2, BBN Tehnologies, 2003.

[21] D. D. Deavours, G. Clark, T. Courtney, D. Daly, S. Derisavi, J. M. Doyle, W. H. Sanders, and P. G. Webster. The Möbius Framework and Its Implementation. *IEEE Transactions on Software Engineering*, 28(10):956-969, October 2002.

[22] Fabrice Stevens. Validation of an Intrusion-Tolerant Information System Using Probabilistic Modeling. Master's thesis, University of Illinois at Urbana-Champaign, 2004.

AUTHOR BIOGRAPHIES

SANKALP SINGH is a Ph.D. student in the Department of Computer Science at the University of Illinois, Urbana-Champaign. His research interests include security quantification and validation. He received his B.Tech. in Computer Science and Engineering from the Indian Institute of Technology, Kanpur in 2001 and his M.S. in Computer Science from the University of Illinois in 2003. His email address is sankalps@crhc.uiuc.edu.

ADNAN AGBARIA received his B.A, M.A, and Ph.D. degrees in 1994, 1997, and 2002 respectively. Both the B.A. and M.A. are in Mathematics and Computer Science, from Haifa University, and the Ph.D. is in computer science from Technion - Israel Institute of Technology. Currently, he works as a postdoctoral research associate in the Coordinated Science Laboratory, University of Illinois at Urbana-Champaign. His research interests include fault-tolerant and intrusion-tolerant computing, network security, parallel and distributed computing, and computer networking. He is also a member of the ACM and IEEE. His email address is adnan@crhc.uiuc.edu.

FABRICE STEVENS received his B.S. in Computer Engineering from ENST (Ecole Nationale Superieure des Telecommunications, Paris, France) in 2002, and his M.S. in Computer Engineering from the University of Illinois at Urbana-Champaign in 2004. He is currently working on network security at France Telecom Research and Development. His email address is fabrice.stevens@francetelecom.com.

TOD COURTNEY received his Bachelor degree in Computer Engineering in 1994 and his Masters in Electrical Engineering in 1996, both from the University of Illinois. He is currently a researcher at the University of Illinois. His interests include performance, dependability, and security system evaluation, as well as stochastic modeling algorithms and software. His email address is tod@crhc.uiuc.edu.

JOHN F. MEYER is a Professor Emeritus in the Department of Electrical Engineering and Computer Science at the University of Michigan, Ann Arbor, Michigan. He has been active in computer and systems research for over 45 years, resulting in a wide variety of publications concerning the areas of fault-tolerant computing and system evaluation. He joined the Michigan faculty in 1967 and, in addition to his university appointments, has held visiting research appointments at laboratories in England, France, Italy, Japan, and Sweden. Prior to 1967, he was a Research Engineer at the California Institute of Technology Jet Propulsion Laboratory, where his contributions included the first patent issued to the National Aeronautics and Space Administration. His current research interests concern the development and application of models for evaluating the performability and survivability of distributed computer and communication systems. He is a Life Fellow of the IEEE and has served the IEEE Computer Society in various capacities, including Chair of the Technical Committee on Fault-Tolerant Computing and membership on the Society's Board of Governors. He is also active in IFIP and has received the Silver Core

from their General Assembly for his services to Working Group 10.4. His email address is jfm@eecs.umich.edu.

WILLIAM H. SANDERS is a Donald Bigger Willett Professor of Engineering in the Department of Electrical and Computer Engineering and the Coordinated Science Laboratory at the University of Illinois. He is the Director of the Information Trust Institute (ITI) at the University of Illinois. He is serving as the Vice-Chair of IFIP Working Group 10.4 on Dependable Computing. In addition, he serves on the editorial board of *Performance Evaluation* and is the Area Editor for Simulation and Modeling of Computer Systems for the *ACM Transactions on Modeling and Computer Simulation*. He is a past Chair of the IEEE Technical Committee on Fault-Tolerant Computing. He is a Fellow of the IEEE and the ACM. Dr. Sanders's research interests include performance/dependability evaluation, dependable computing, and reliable distributed systems. He has published more than 160 technical papers in those areas. He served as the General Chair of the 2003 Illinois International Multiconference on Measurement, Modelling, and Evaluation of Computer-Communication Systems. He has served as co-Chair of the program committees of the 29th International Symposium on Fault-Tolerant Computing (FTCS-29), the Sixth IFIP Working Conference on Dependable Computing for Critical Applications, Sigmetrics 2003, PNPM 2003, and Performance Tools 2003, and has served on the program committees of numerous conferences and workshops. He is a co-developer of three tools for assessing the performability of systems represented as stochastic activity networks: METASAN, *UltraSAN*, and Möbius. Möbius and *UltraSAN* have been distributed widely to industry and academia; more than 300 licenses for the tools have been issued to universities, companies, and NASA for evaluating the performance, dependability, security, and performability of a variety of systems. He is also a co-developer of the Loki distributed system fault injector and the AQuA/ITUA middlewares for providing dependability/security to distributed and networked applications. His email address is whs@crhc.uiuc.edu.

PARTHA PAL is a division scientist at BBN Technlogies, and he leads the survivability research thrust in the Distributed Systems Advanced Middleware Technology group. He is currently working on the architecture, design, implementation, and evaluation of a pathfinder system for the U.S. Dept. of Defense. He is a senior member of the IEEE. His email address is ppal@bbn.com.