Model-Based Validation of an Intrusion-Tolerant Information System*

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

†ECE Department
University of Illinois
{fsteven1, tod, adnan, sankalps, whs}@crhc.uiuc.edu

††EECS Department
University of Michigan
jfm@umich.edu

‡BBN Technologies
Cambridge, MA 02138
ppal@bbn.com

Abstract

An increasing number of computer systems are designed to be distributed across both local and wide-area networks, performing a multitude of critical information-sharing and computational tasks. Malicious attacks on such systems are a growing concern, where attackers typically seek to degrade quality of service by intrusions that exploit vulnerabilities in networks, operating systems, and application software. Accordingly, designers are seeking improved techniques for validating such systems with respect to specified survivability requirements. In this regard, we describe a model-based validation effort that was undertaken as part of a unified approach to validating a networked intrusion-tolerant information system. Model-based results were used to guide the system's design as well as to determine whether a given survivability requirement was satisfied.

1. Introduction

Numerous successful attacks into information systems have illustrated the fact that complete security is difficult to achieve. As more software and systems are introduced, more vulnerabilities are created. Intrusion prevention does not appear to be sufficient to deal with all attacks. *Intrusion tolerance* is a growing area of computer security that can be used to design more reliable and survivable systems. Intrusion tolerance is an approach for handling malicious attacks [2, 4, 5]. Combined with security, it can constitute "defense-in-depth," enforcing the attacker to spend more effort and time to break into a system.

No methodology so far can claim to prove a certain degree of security in a system. In fact, most attempts at validation of security have been non-quantitative. However, as mentioned above, it is practically impossible to build a perfectly secure system. Hence, it is very important to be able to quantitatively validate the efficacy of secure systems in general, and intrusion-tolerant systems in particular. Efforts for quantitative validation of security have usually been based on formal methods [9], or have been informal, using "red teams" to try and to compromise a system [11]. Both approaches, while being valuable in identifying system vulnerabilities, have their limitations, especially when they are applied to large intrusion-tolerant systems.

Probabilistic modeling has been receiving increasing attention as a mechanism to validate security. It is especially suited to intrusion-tolerant systems, since by definition, intrusion tolerance is a quantitative and probabilistic property of a system. Any probabilistic model for validating 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. [10]. 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 [8] 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 uncertainities therein. Madan et al. [12] 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

*This paper was supported by DARPA contract number F30602-02-C-0134.

state of the system in terms of high-level events that may lead to failures. Sheyner et al. [15] have tried to build attack trees automatically using formal methods, and then analyze those trees using Bayesian networks. Ortalo et al. [13] 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. [16] 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. [7] have used a similar approach to evaluate the security and performance of several intrusion-tolerant server architectures.

In this paper, we use a probabilistic model for validating an intrusion-tolerant system that combines intrusion tolerance and security. The probabilistic model makes use of 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 use the probabilistic modeling to compare different design configurations, allowing the designers of the system to make choices that maximize the intrusion tolerance provided by the system before they actually implement the system. Lastly, we use the model to prove that the system would meet a set of quantitative survivability requirements.

The system we use in this paper is a networked information system called a *Joint Battlespace Infosphere* (JBI). JBI is currently being developed by agencies and companies working for the United States government. A JBI will serve as a substrate for integrating current/planned command and control systems. As envisioned in [1], a JBI provides individual users with specific information required for their functional responsibilities during a crisis or conflict. Supporting this capability is a JBI *core* consisting of networked computers that implement protocols, processes, and common core functions. In particular, the basic core-provided services of *publish*, *subscribe*, and *query* (PS&Q) permit JBI *clients* (mission applications) to exchange information in the form of *information objects* (IOs).

When deployed, a JBI (both core and clients) must be *survivable* in the sense that it is capable of fulfilling its mission in the presence of attacks, failures, or accidents (see [6], for example). Although the nature of specific JBI missions will vary quite widely, any mission will require a JBI implementation that satisfies stringent, quantitative survivability requirements, particularly with respect to the PS&Q services.

The remainder of this paper is organized as follows. Section 2 provides an overview of the distributed system design studied. Section 3 presents the requirement for correct system functionality. Section 4 describes the attack model developed for the validation effort. Section 5 discusses details of the probabilistic model implementation. Section 6 presents results obtained from the models. We conclude our work in Section 7.

2. Overview of the IT-JBI Design

The IT-JBI system consists of multiple clients communicating with each other through a central core, as shown in Figure 1. The core consists of three *zones* (or layers) of components: the *crumple zone* (outermost layer), the *operations zone*, and the *executive 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 specific connections using 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/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 mission-specific and domain-specific knowledge to identify corrup-

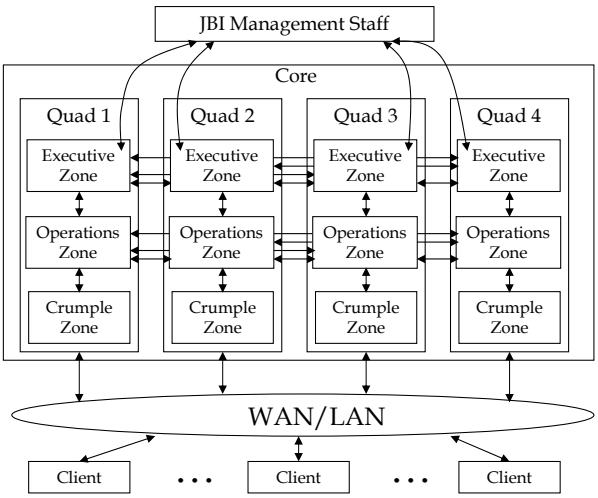

Figure 1. The IT-JBI Architecture

tion within the contents of the IO.

Security monitoring and maintenance are performed by local components distributed across each host in the client, the crumple zone, the operations zone, and a centralized *correlator* (Co) in the operations zone. The components local to each host 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-JBI. It monitors intrusion alerts received from the correlator and generates commands for the appropriate response to counter the intrusion. Human operators monitor the IT-JBI via displays generated by the SM, and can manually initiate specific responses from the SM.

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

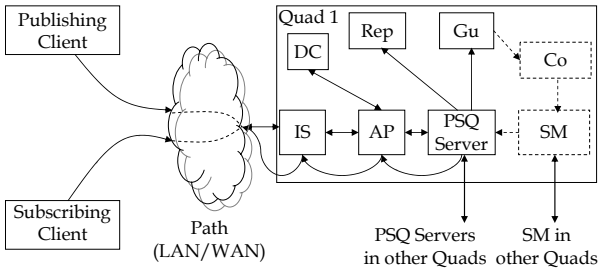

Figure 2. Publish Data Flow

hosts within a single quadrant of the core.

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 Figure 2, is the data flow among the IT-JBI 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.

The second data flow of interest is the alert/response data flow associated with the intrusion detection/tolerance capability 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 Figure 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

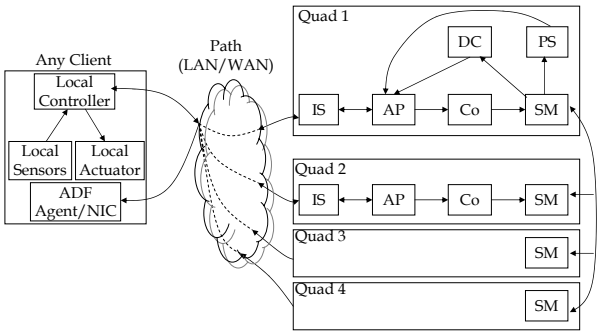

Figure 3. Alert/Response Data Flow of a Client

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. Publish Requirement

The core-provided services of publish, subscribe, and query are critical to fulfilling any JBI mission. Accordingly, IT-JBI survivability with respect to these services is a dominant concern of the validation process. In the case of publish, the corresponding survivability requirement can be stated as

PUB: Assuming precondition C_{PUB}, a client's request to publish an IO is processed successfully with probability at least p_{PUB} ($0 < p_{PUB} < 1$),

where, in terms of the IT-JBI, C_{PUB} and the event

E_{PUB} = a client's request to publish an IO is processed successfully

are defined more precisely as follows (requirements for subscribe and query can be formulated in a similar manner).

C_{PUB} is the conjunction of the following preconditions, i.e., events that are assumed to have occurred prior to a client's publish request. In an expanded validation study, it would be possible to remove one or more of these conditions by either proving that they were satisfied (if they were not quantitative) or inserting them as additional events that qualify the meaning of "processed successfully."

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

C^2_{PUB} = the publishing client's mission application interacts with the client as intended (including provision of adequate and accurate metadata).

E_{PUB} 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).

Accordingly, PUB can be restated more concisely as

PUB: $P[E_{PUB}|C_{PUB}] \geq p_{PUB}$, where $P[E_{PUB}|C_{PUB}]$ is the conditional probability (relative to condition C_{PUB}) of E_{PUB} occurring with respect to a randomly chosen publish request during the mission duration.

It is obvious that the determination of whether PUB is satisfied by the IT-JBI, given some specified value of p_{PUB} (i.e., whether PUB is true when so instantiated), reduces to an evaluation of the success probability $P[E_{PUB}|C_{PUB}]$ and a comparison of the result with the value of p_{PUB}. Model-based evaluation is employed for that purpose.

4. Description of the Attack Model

This section presents a detailed description of how attacks and resulting intrusions are represented in the model for the publish service. 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.

4.1. 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-JBI 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.

4.2. 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 Section 4.1). Define $MTTE$ to be the mean time between successive exploitations of a known vulnerability. The typical value of $MTTE$ in our study 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-JBI.

4.3. 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-JBI (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 system.

- *Data-level attacks* exploit a vulnerability in an IT-JBI application on the targeted component. 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-JBI 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-JBI protocol packets, signed with signatures from host-specific private keys.

Figure 4 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 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.

4.4. 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).

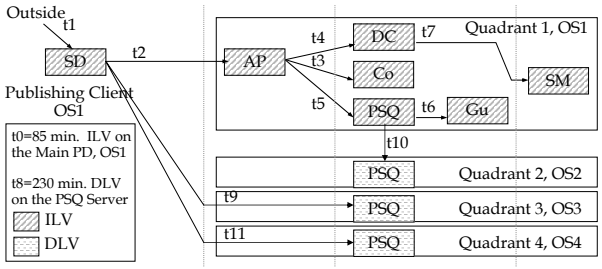

Figure 4. Example of Attack Propagation

- *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.

4.5. 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.

5. Model Overview

The integrated validation procedure consists of many techniques for validation, including formal methods, experimentation, and probabilistic modeling [14]. The discussion that follows focuses on the probabilistic model used for validation of the IT-JBI design with respect to the publish requirement PUB. This probabilistic model was accomplished by constructing models representing the components of the IT-JBI system. The models were built with the stochastic activity network (SAN) formalism using the Möbius tool [3]. Eighteen different SAN models were built to represent the various components of the system. The models were combined using the Rep-Join composition formalism to construct the system model. In the Rep-Join formalism, *Join* nodes are used to connect together multiple heterogeneous submodels and allow them to share state variables. *Rep* nodes are a special case of Join nodes, and are used to create multiple copies of a single submodel.

Four main components are used in the SAN formalism. *Places* represent the state of the system and may hold *tokens*. *Activities* generate events that change the state of the system and represent delays in the system. Activities with *cases* represent probabilistic choices among multiple possible outcomes. *Input gates* give specific and detailed definitions of conditions necessary for activities to be enabled (able to execute). *Output gates* specify detailed and possibly complex changes to the system after an activity executes. Counted individually, the eighteen IT-JBI detailed models constructed for our case study contain 787 places, 578 activities, 253 input gates, and 354 output gates. The system model, formed by combining and replicating the detailed component models, has 10,771 places, 7,850 activities, 3,370 input gates, and 4,471 output gates. Space does not permit a detailed description of the entire model, but the remainder of this section consists of an overview of some significant features of the model. For more details, the reader can refer to [17].

5.1. System Model

The high-level model for the IT-JBI system (Figure 5) consists of two clients (Join *PubClient* and *SubClient*) communicating with the core (Rep *Core*) through the network (submodel *Path*). Multiple processes are running on each client, represented by four submodels under each client Join. The processes on the clients include the primary process on the host, the sensors, the actuator, and the local controller (for the publishing client, the processes are *PubClient_main*, *Pub_Se*, *Pub_Ac*, and *Pub_LC*, respectively). The fifth submodel under each of those Joins implements the attack propagation model, and was described in Section 4. The two remaining submodels, *measures* and *attack_discovery*, implement some measures and discovery of new vulnerabilities, respectively.

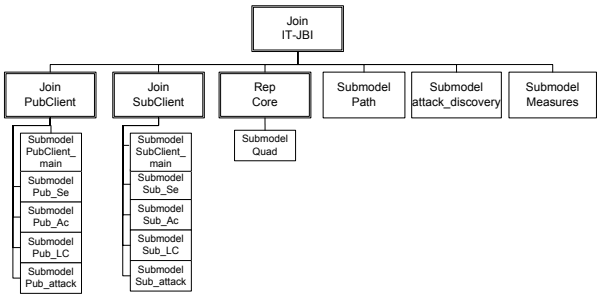

Figure 5. Top-Level Composed Model

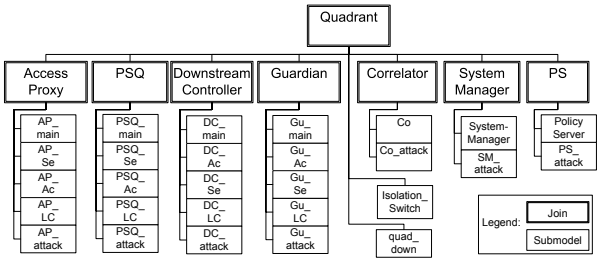

Figure 6. Quad Composed Model

The core is a replica of four quadrants (submodel *Quad1*), detailed in Figure 6. Each quadrant is a Join of several submodels (*Access Proxy*, *PSQ*, *Downstream-Controller*, *Guardian*, *Correlator*, *PS*, *SM*, and *Isolation_Switch*) that implement the core components. The access proxy, PSQ server, downstream controller, and guardian have an Intrusion Detection System (IDS). Therefore, their respective Joins have IDS submodels (for instance, *AP_Se*, *AP_Ac*, and *AP_LC*). The correlator, policy server, system manager, and quadrant isolation switch don't have any IDS components.

5.2. Attack Model Implementation

The attack model is represented by multiple submodels within the system model. The *attack_discovery* submodel, under the *IT_JBI* Join in Figure 5, implements the functionality for the discovery of new vulnerabilities in the system, and identifies which process of which component on which operating system will be affected. Central to the *attack_discovery* model is an activity that generates a newly discovered vulnerability every $MTTD$ minutes on average. Activities with cases are then used to probabilistically determine specific characteristics about the vulnerability, such as the type of operating system it affects.

Each host in the system also has a submodel called *attack_propagation* (represented in Figures 5 and 6 by *Sub_attack*, *PSQ_attack*, *DC_attack*, etc) associated with it.

The model represents the exploitation of known vulnerabilities for that host and the potential propagation of successful attacks from a compromised host to possible network neighbors. The model stores information in places that represent the state of the host, such as whether each process on the host is crashed/down, compromised by a successful attack, or operating normally. For the processes that are compromised, activities with detailed output gates are used to encode the list of connected processes and hosts on which the attack could propagate. Each process on each host in the system has a place that records the number of other processes that are currently compromised and able to launch attacks on the process. Such places are combined appropriately with the corresponding places in other attack models in the system, through definition of shared variables at the multiple Rep/Join levels of the composed model.

An example of one such place is DA_PSQ. DA_PSQ is initialized to the value 0 (for each of the quadrants), as no outsider can start a data attack directly on the PSQ. If the publishing client becomes compromised at time t_0, DA_PSQ (for each quadrant) increases to 1, as the client can be used to launch a data attack on any PSQ server. However, as long as no data-level vulnerability has been found, the PSQ server won't be attacked. Assuming that at time t_1 such a vulnerability is found, the time before each PSQ server is compromised is given by an exponential distribution with mean $MTTE$.

6. Design Validation Results

The probabilistic model described in the previous sections represents highly aggressive attackers' attempts to intrude into the IT-JBI system while the system is performing its publish functionality. The model is used to give evidence that the publish requirement is met, as well as 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. There are also 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 ($MTTE$). Varying these parameters changes the aggressiveness of the attacker.

Each variation of these input parameters defines a unique configuration of the system and attack model. The measures were obtained by discrete-event simulation, and by gathering statistical results. In this section, we present the results for three of the variations studied to date, specifically those derived by varying the vulnerability discovery rate, the degree of operating system diversity among the four quadrants

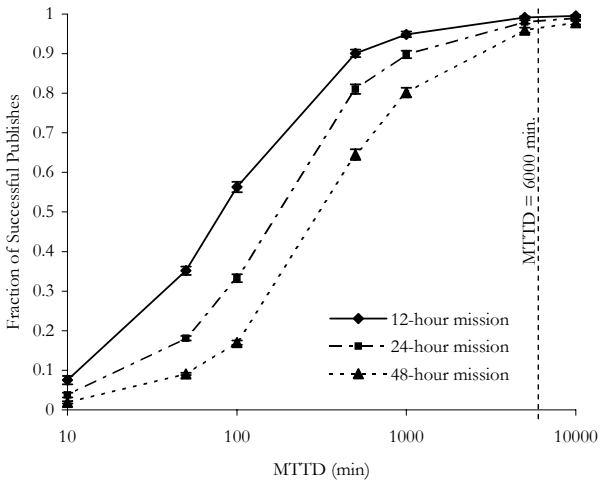

Figure 7. $P[\mathbf{E}_{\text{PUB}}|\mathbf{C}_{\text{PUB}}]$ versus MTTD: Default Configuration of the IT-JBI

in the IT-JBI core, and the Autonomous Distributed Firewall Network Interface Card (ADF NIC) policies.

6.1. Vulnerability Discovery Rate

Our first experiment illustrates how the IT-JBI meets the publish requirements described in Section 3. Based on the system requirements, we chose a value of 0.95 for p_{PUB}.

The sequence of responses to repeated attack on a host are 1) restart of the compromised process (Type 1), 2) secure reboot of the host (Type 2) for the next three intrusions on the host, and then 3) isolation (Type 3). The IT-JBI runs four different operating systems, one within each quadrant. One of the four is *traditional*, and the other three are *secure*. Compromised sensors do not detect any further intrusion. Any component creating or forwarding alerts or commands (to restart a process, reboot a host, or isolate a host) will drop those alerts or commands.

The evaluation of $P[\mathrm{E}_{\text{PUB}}|\mathrm{C}_{\text{PUB}}]$ was based on the following values for the model parameters. $MTTE$ is set to 2 minutes. When an intrusion occurs, the effect is to compromise the host with probability 1.0, crash it with probability 0, and lead to a Denial of Service with probability 0. When vulnerabilities are found, the probability that they will affect all operating systems (common mode) is 0.4. When the sensors are not compromised, they detect intrusions with a probability of 0.95, after an exponential time averaging 2 seconds. IDS components send heartbeats every minute. Finally, the attacker can perform all types of attacks (infrastructure-level, data-level, and across process domains).

A study presented in [14] estimated the value of $MTTD$ to be 6,000 minutes, by counting the number of vulnerabil-

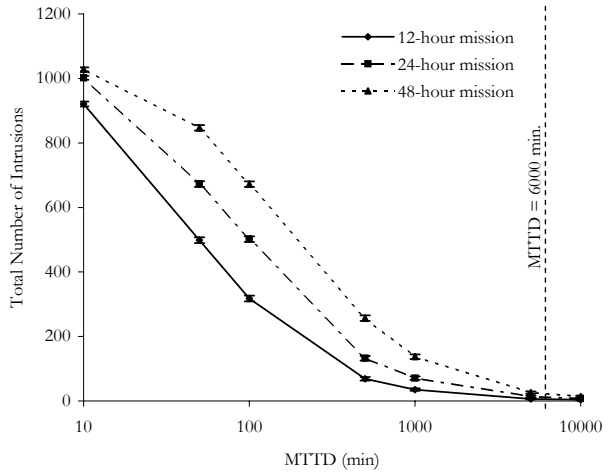

Figure 8. Number of Intrusions versus MTTD: Default Configuration of the IT-JBI

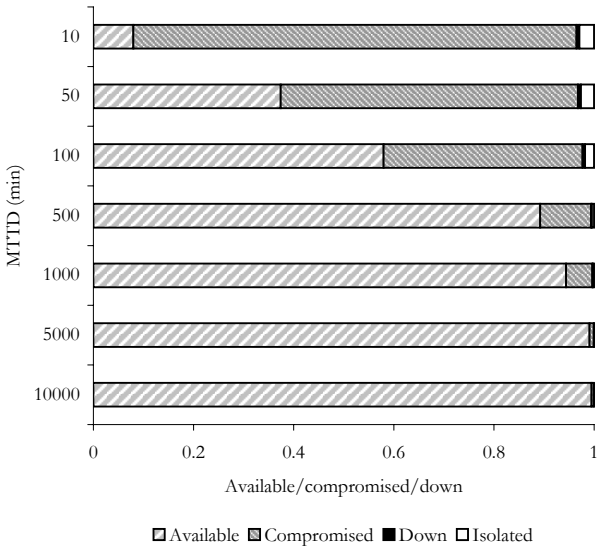

Figure 9. Availability of the AP versus MTTD: Default Configuration of the IT-JBI

ities over a given time period that the system would have been vulnerable to. Given the uncertainty about the exact value of $MTTD$, we decided to present the results in this paper with an $MTTD$ varied widely, typically between 100 and 10,000 minutes. The design decisions were made using the value of 6,000 minutes for $MTTD$.

Figures 7 and 8 show the effect that varying the vulnerability discovery rate has on the variation of $P[\mathrm{E}_{\mathrm{PUB}}|\mathrm{C}_{\mathrm{PUB}}]$ and the number of intrusions, respectively. Three mission durations are considered: 12 hours (720 minutes), 24 hours (1,440 minutes), and 48 hours (2,880 minutes).

The results show that the IT-JBI provides correct functionality with high probability (95% of all publishes are successful during a 12-hour mission) when new vulnerabilities are discovered once a day or less often. Figure 9 presents the availability of the access proxies for a 12-hour mission, versus the vulnerability discovery rate. The other components in the IT-JBI present very similar figures: the availability of each component of the IT-JBI is high (0.94 or more for a 12-hour mission) when new vulnerabilities are discovered, on average, no more than once every 16 hours (1,000 minutes).

6.2. 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 IT-JBI.
- *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.

The graph presented in Figure 10 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 $P[\mathrm{E}_{\mathrm{PUB}}|\mathrm{C}_{\mathrm{PUB}}]$ versus the vulnerability discovery rate. Diversity significantly increases the performance of the design: when $MTTD = 200$ minutes, $P[\mathrm{E}_{\mathrm{PUB}}|\mathrm{C}_{\mathrm{PUB}}]$ 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

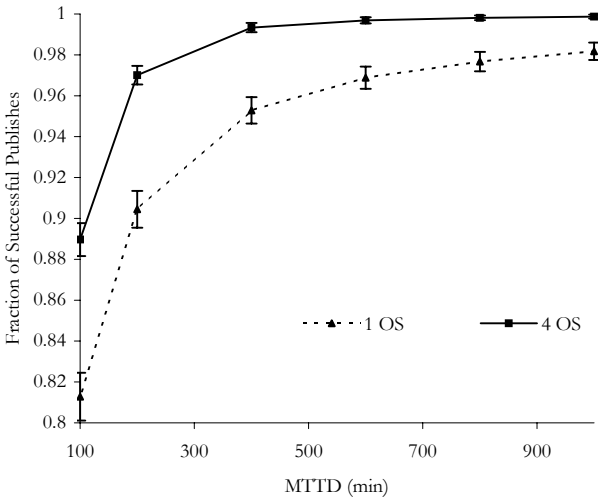

Figure 10. $P[\mathbf{E}_{\text{PUB}}|\mathbf{C}_{\text{PUB}}]$ **versus MTTD: 4 OS vs 1 OS**

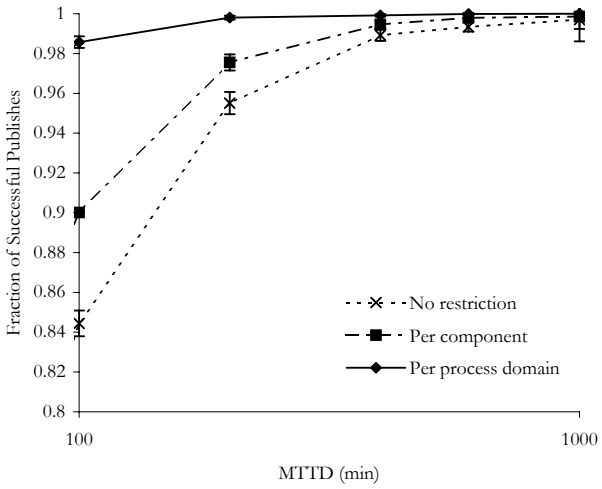

Figure 11. $P[\mathbf{E}_{\text{PUB}}|\mathbf{C}_{\text{PUB}}]$ **versus MTTD: ADF Policy Study**

semantic checks done on the access proxy for any incoming traffic to the core.

6.3. ADF NIC Policies

ADF NICs are local firewalls on each component, administrated 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.

Figure 11 presents the results reflecting the three configurations. The graphs reveal that the per-process-domain policy is by far the best of all three: for $MTTD = 100$ minutes, $P[\text{E}_{\text{PUB}}|\text{C}_{\text{PUB}}] = 84.4\%$ 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 rule-set 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.

7. Conclusions

We described the infrastructure of a networked information system that serves as a substrate for integrating current and planned command and control systems. Stochastic models of the system and of the attacker were presented. Attacks were classified as infrastructure-level, data-level, or across-process-domain, and attack effects were classified as compromise, crash, or DoS. We conducted model-based experiments that evaluated the survivability of the system when stressed by those 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.

This paper illustrates how probabilistic modeling can be used in an integrated validation procedure and successfully bring insight and feedback on a design. It allows us to compare different algorithms, features, or infrastructures. Moreover, building such a model required a detailed specification of the architecture; therefore, the construction of the model resulted in an enhancement of the level of detail in the design specifications.

8. Acknowledgments

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 DARPA, particularly Jay Lala and Lee Badger, and 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.

References

[1] U. A. F. S. A. Board. Report on Building the Joint Battlespace Infosphere, Tech. Report SAB-TR-99-02, 1999. http://www.sab.hq.af.mil/archives/reports/index.htm.

[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] 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.

[4] 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.

[5] 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.

[6] R. Ellison, D. 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, CERT Coordination Center, Carnegie Mellon University, November 1997.

[7] 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.

[8] 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.

[9] C. Landwehr. Formal Models for Computer Security. *Computer Surveys*, 13(3):247–278, Sept. 1981.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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

[15] 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.

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

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