Probabilistic Quantification of Survivability Properties

William H. Sanders

(Based on work with Adnan Agbaria, Tod Courtney, Michel Cukier, John Meyer, Sankalp Singh, Fabrice Stevens, Franklin Webber, among others)

University of Illinois at Urbana-Champaign
Urbana, Illinois 61801
http://www.crhc.uiuc.edu/PERFORM/

Workshop on Measuring Assurance in Cyber Space
IFIP Working Group 10.4 – Monterey, CA, 2003
June, 2003

This research has been supported by DARPA contract F30602-00-C-0172

Motivation

- Intrusion tolerance is an approach to security that aims to increase the likelihood that an application will be able to continue to operate correctly in spite of malicious attacks that may occur and may result in successful intrusions.
- Before intrusion tolerance can be accepted as an approach to providing security, techniques must be developed to validate its efficacy.
- Validation should be done:
  - During all phases of the design process, to make design choices
  - During testing, deployment, operation, and maintenance, to gain confidence that the “amount” of intrusion tolerance provided is as advertised.

Existing Validation Approaches

- Most traditional approaches to security validation have focused on avoiding intrusions (non-circumventability), or have not been quantitative, instead focusing on and specifying procedures that should be followed during the design of a system (e.g., the Security Evaluation Criteria [DOD85, ISO99]).
- When quantitative methods have been used, they have typically either been based on formal methods (e.g., [Lan81]), aiming to prove that certain security properties hold given a specified set of assumptions, or been quite informal, using a team of experts (often called a “red team,” e.g. [Low01]) to try to compromise a system.
- Both of these approaches have been valuable in identifying system vulnerabilities, but probabilistic techniques are also needed.

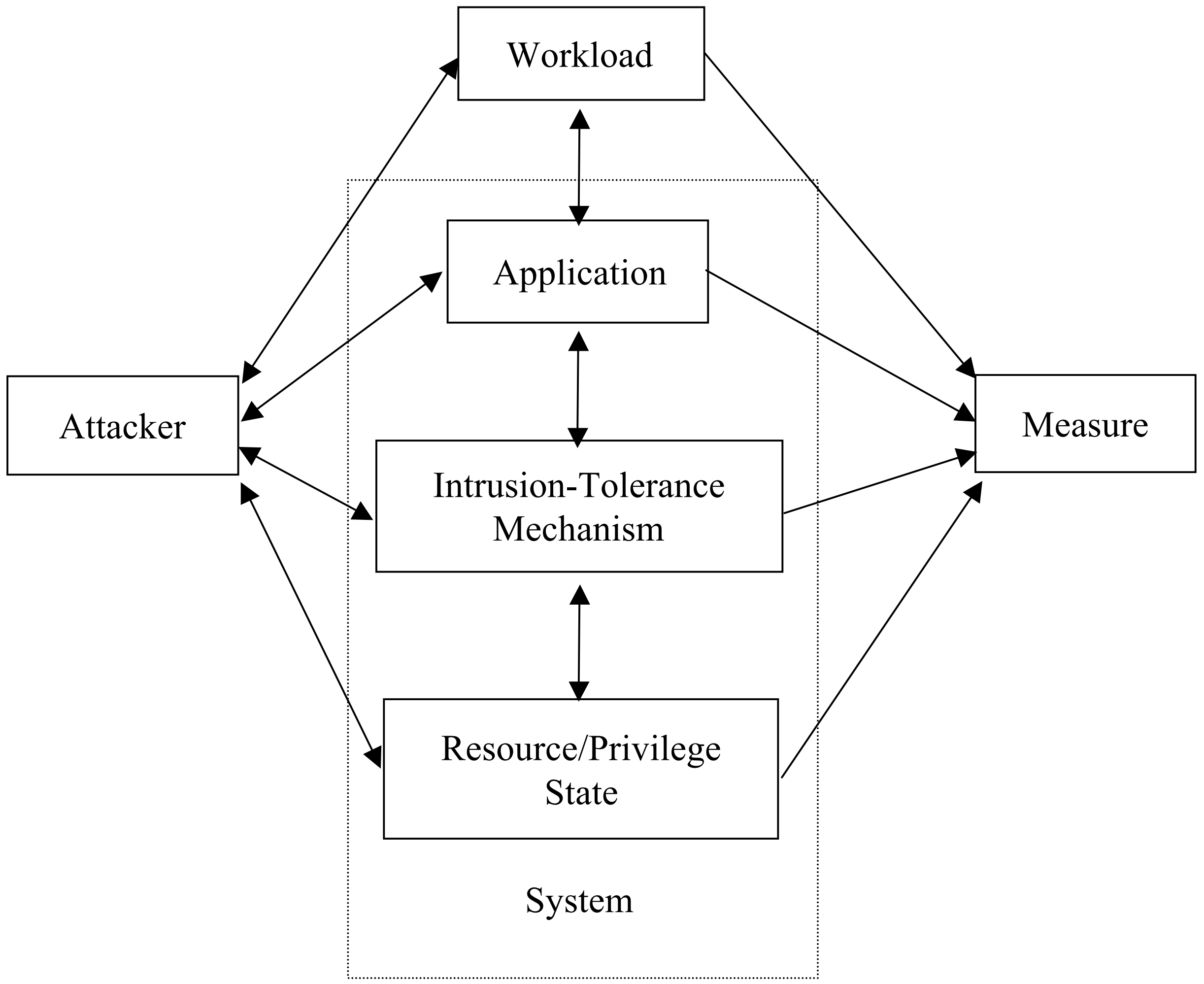
Related Probabilistic Security Validation Work

- Several attempts have been made to use probabilistic methods for assessing security:
  - Littlewood et al., 1993: identified questions that need to be answered to make probabilistic security evaluation viable
  - Jonsson et al., 1997: built quantitative model of intrusion based on attacker behavior (the only source of uncertainity considered)
  - Gong et al., 2001: 9-state model of an intrusion tolerant system
  - Jha and Wing, 2001, and Ortalo et al., 1999: constructed scenario graphs for modeling known system vulnerabilities
- These approaches provide a good starting point. They show that:
  - measures similar to those used in dependability evaluation can be defined;
  - it may be possible to model attackers; and
  - the system can be represented using state-level models, capturing either known or unknown vulnerabilities

Probabilistic Quantification of Security Issues

- *The effect of the attacker must be modeled*, either explicitly or logically. Focus should not be only on the worst-case scenario (strongest possible attack), but the distribution of possible attacks
- *Submodels* of the intrusion tolerance mechanism being used, the application being defended, and the resource/privilege state of the system should be created
- *Probability* can enter the models through: a) inherent random elements in the attacks and defense that make them unpredictable, and b) abstraction the parts of the system, especially the vulnerabilities and how they are exploited
- *Appropriate level of detail/abstraction* should be used:
  - System submodels should represent the parts of the system that are important relative to the type of attacks considered, and the expression of the particular survivability measure
  - Attacker submodel may either represent details of the intrusion itself or represent the *effect* of the intrusion

Possible Probabilistic Validation Framework



Our Approach

- Probabilistic modeling using Stochastic Activity Networks (SANs) of attacker effects and intrusion-tolerant mechanism
- Demonstration of approach by using SANs to model and validate an intrusion-tolerant replication system, that is a part of the Intrusion Tolerance by Unpredictable Adaptation (ITUA) architecture
- Modular construction of model for easy adaptation to other intrusion-tolerant systems
- Definition of several measures on the model to characterize the level of intrusion tolerance provided by the system, and provide insights into the relative merits of various design choices by studying variations in the measures in response to change in system (and attack) parameters

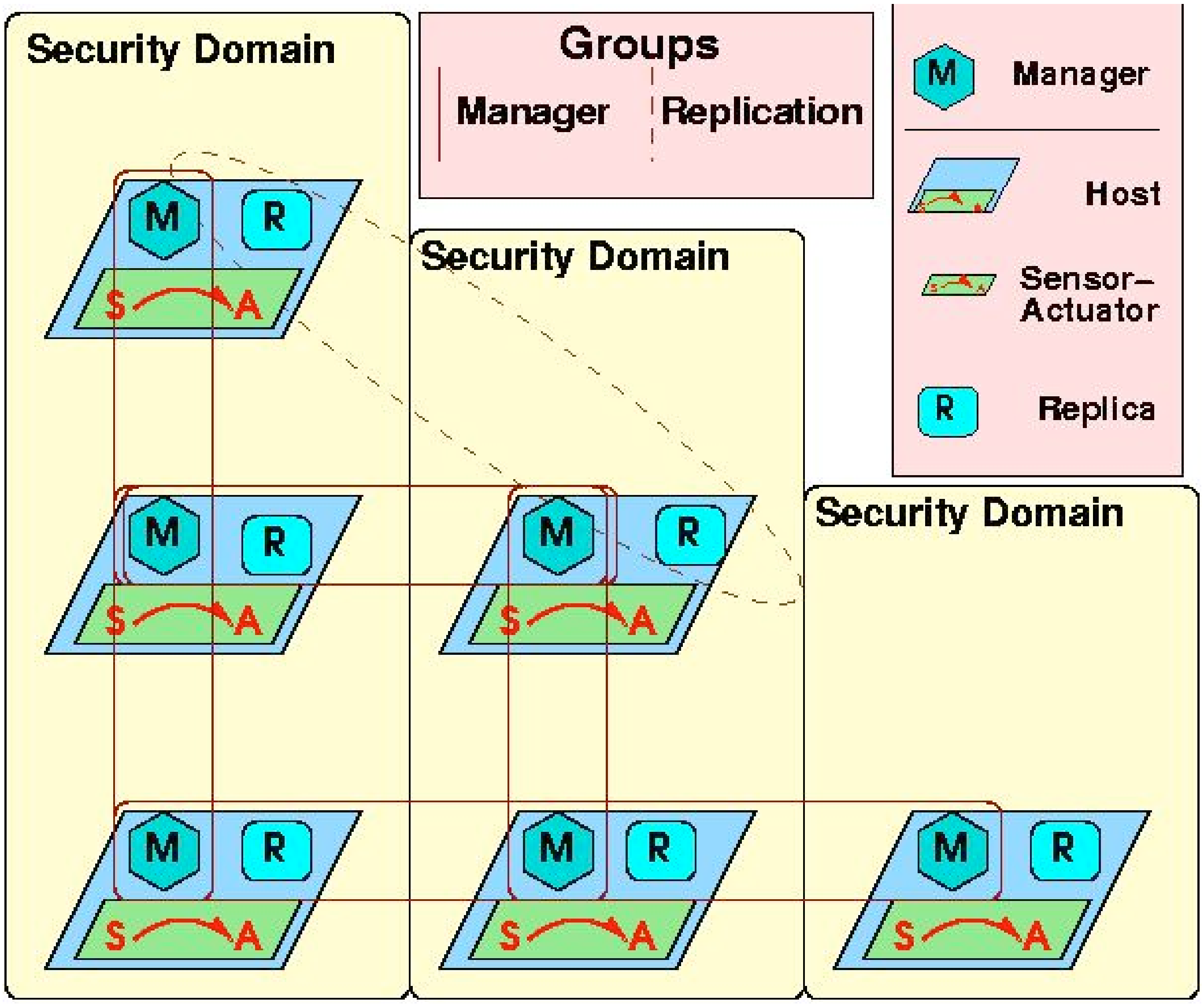
Overview of ITUA Approach to Intrusion Tolerance

- *I*ntrusion *T*olerance by Unpredictable *A*daptation (ITUA) architecture is a middleware-based intrusion tolerance approach that helps applications survive certain kinds of attacks.
- Uses an intrusion-tolerant group communication system
- Integrates a set of COTS security tools, that together with the IT-GCS, to detect corrupt processes
- Provides a decentralized replica management facility that responds (possibly in an unpredictable way) to intrusions. The system deals with arbitrary failures of replicas and management entities.

ITUA Architecture

- System is divided into multiple security domains, each with one or more hosts, implementing an attacker boundary
- Managers: Each host runs a manager. Managers:
  - Are part of a communication group (manager group). IT-GCS used to multicast messages within a group
  - Determine the group structure of managers and replicas
  - Propagate information regarding important state changes
  - Convict corrupt members of the system to prevent known processes from corrupting the system
- Application replicas: Application objects are replicated by the middleware and distributed across security domains.
  - Any number of applications allowed
  - A security domain can have only one replica from each application
  - All replicas of an application group form a group

Pictorial View



Overview of ITUA Replication System

- Members of group need to reach consensus to:
  - Convict a corrupt member of the group (replication or manager group)
  - Help managers decide where to place a new replica (manager group)
- Any number of members of a group may be in a corrupt, but not yet detected state.
- We assume:
  - Byzantine fault tolerance using authenticated Byzantine agreement under a timed asynchronous environment.
  - (⇒) less than a third of the *currently active* group members can be corrupt and still allow the group to reach a consensus.

Detection of and Recovery from Corrupt Replicas

- By other replicas in its replication group *when* it exhibits corrupt behavior during group communication
  - Detection and conviction by the correct members of the group if less than a third of currently active members corrupt
  - Convicted replica excluded from group communication
  - Each correct replica of the group informs its manager, which (if it is not corrupt itself) propagates the information among the manager group
  - If enough correct managers, they reach a consensus, and pick a random domain (which does not already have a replica of that application) and a random host within the domain to restart the replica
- By the intrusion detection software running on its host
  - Detects intrusions into the host operating system or a replica running on the host
  - Informs the local manager about the detection. Further dissemination and subsequent restart of replica(s) similar to detection by replication group

Preemptive Action of Managers

- Managers take preemptive action by convicting and excluding the security domain (and all the hosts/replicas therein) that had the corrupt replica.
  - Motivation: good chance that attack might have propagated to other hosts in the domain.
  - Replicas killed as a result may have to be restarted in other domains.
- We have also analyzed and compared an alternative approach in which only the host running the corrupt replica is excluded.
- We assume system is left to itself with minimum human intervention:
  - We do not model manual repair of excluded hosts/domains.
  - Hence, system, as modeled, can run out of domains to restart new replicas to replace the killed ones.

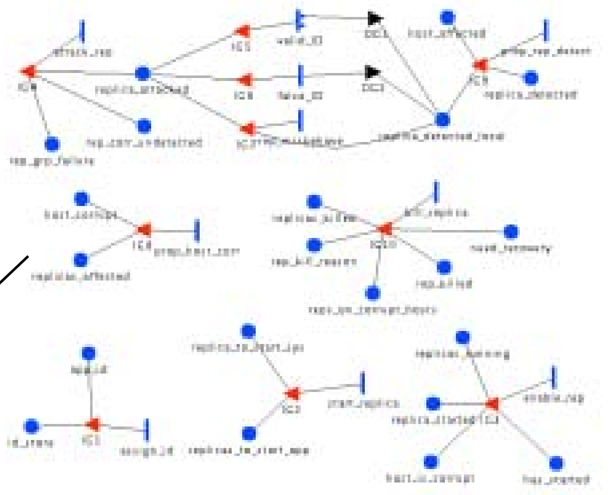
Example Attacker Model

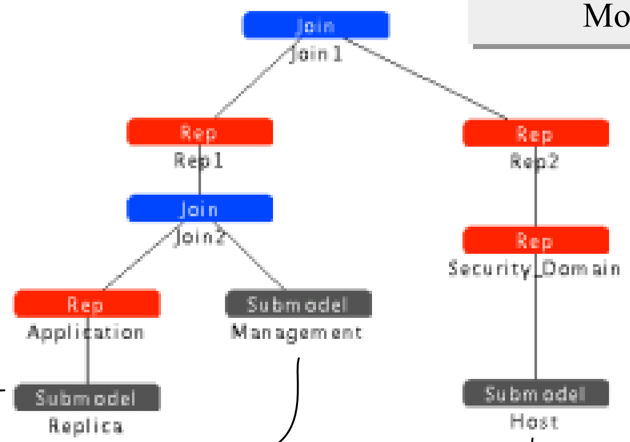
- We model the *effect* of model successful intrusions
- Attackers can target host OS and services, application objects, or even the management infrastructure
- We consider three distinct classes of attacks (Jonsson 1997):
  - **Script-based attacks**: most frequently attempted, fairly high chances of detection
  - **More exploratory attacks**: less frequent, employed by more experienced attackers using combination of tricks and exploits, lesser chances of detection
  - **Innovative attacks**: rare, but difficult to detect
- Attackers learn from successful intrusions:
  - Corruption of a host increases the likelihood of corruption of other hosts in its security domain
  - Intrusion into the host OS greatly increases the chances of intrusion into other assets on the host

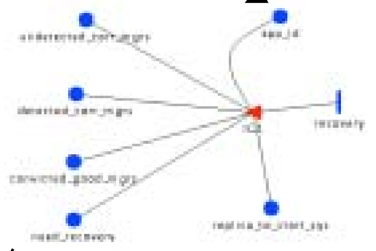
Probabilistic Model Overview

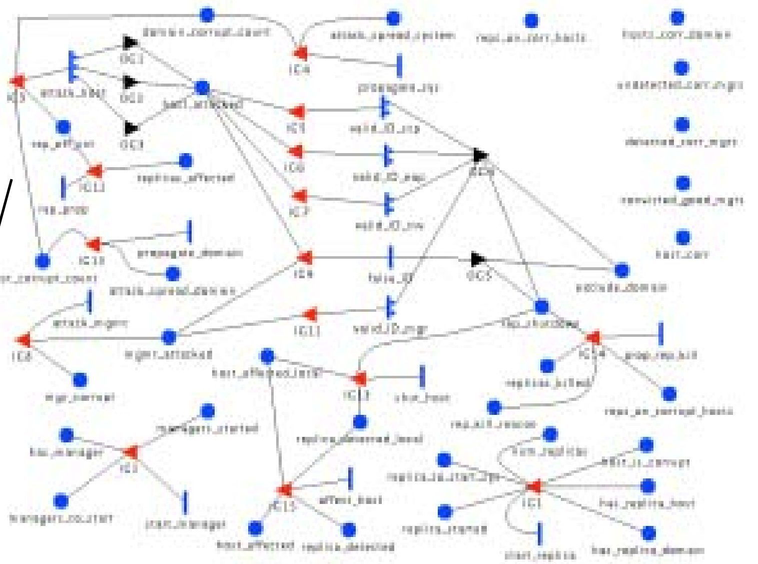
- We use Stochastic Activity Networks (SANs) as our probabilistic modeling formalism.
- Our approach consists of:
  - Construction of atomic SAN models for various components of the architecture: *a host*, *an application replica*, and *the management algorithm.*
  - Composition of the atomic SAN models to construct a SAN model for the overall architecture.
  - Definition of measures in terms of model behavior.
  - Simulation of model behavior to obtain values for measures.

Example Probabilistic Survivability Model







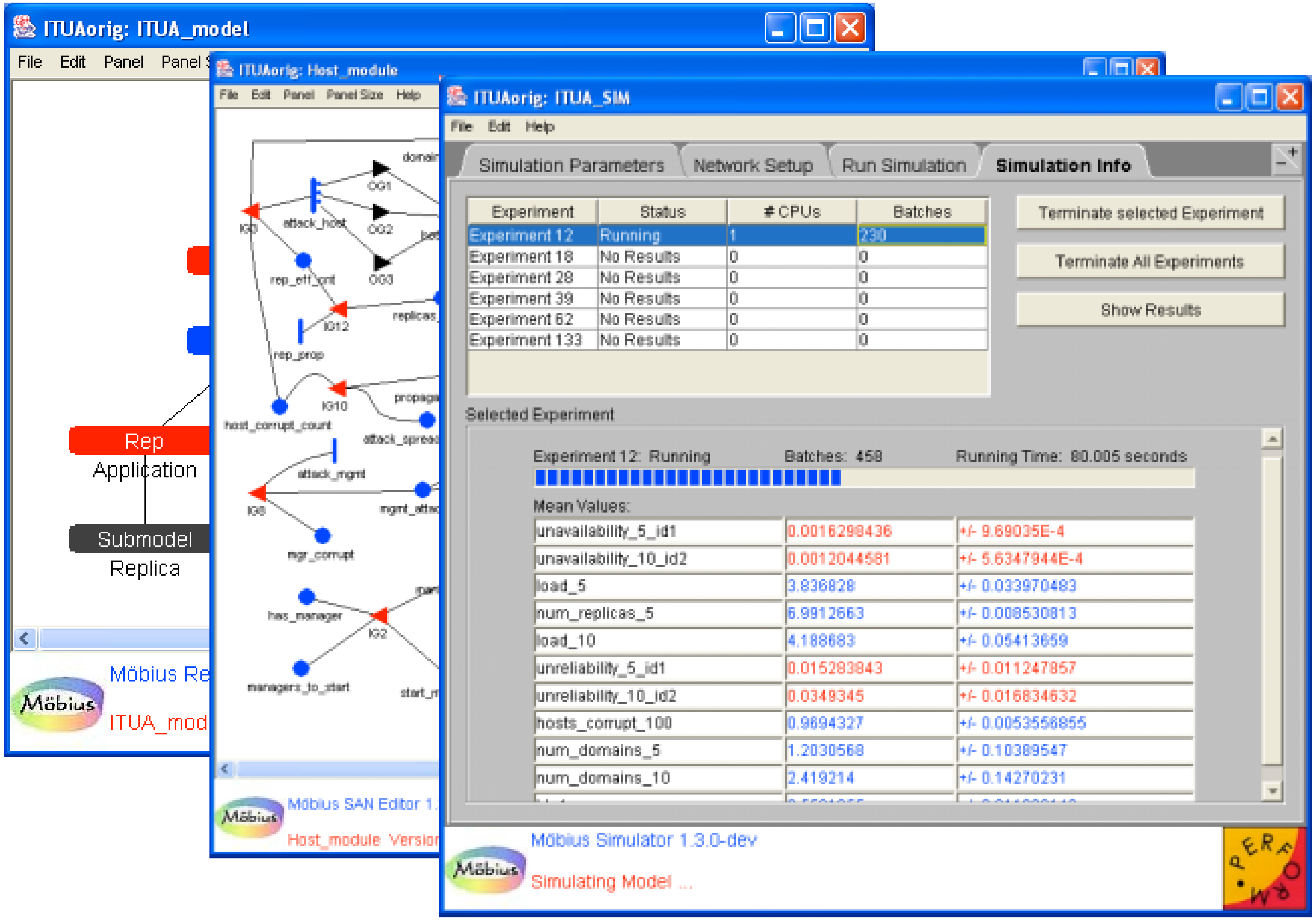


Replica submodel models behavior of a single replica: starting of replica, attacks on replica, detection of infiltration by IDS, misbehavior by infiltrated replicas and its detection by other replicas.

Management submodel models the process of recovery by the management infrastructure through the starting of new replicas

Host submodel models the activities on a single host: attacks on host, detections and false alarms by IDS, starting replicas and management entities, and shutting down

Implementation in Möbius



Experiment	Status	# CPUs	Batches
Experiment 12	Running	1	230
Experiment 18	No Results	0	0
Experiment 28	No Results	0	0
Experiment 39	No Results	0	0
Experiment 62	No Results	0	0
Experiment 133	No Results	0	0

Selected Experiment

Experiment 12: Running Batches: 458 Running Time: 80.005 seconds

Mean Values:

unavailability_5_id1	0.0016298436	+/- 9.69035E-4
unavailability_10_id2	0.0012044581	+/- 5.6347944E-4
load_5	3.836828	+/- 0.033970483
num_replicas_5	6.9912663	+/- 0.008530813
load_10	4.188683	+/- 0.05413659
unreliability_5_id1	0.015283843	+/- 0.011247857
unreliability_10_id2	0.0349345	+/- 0.016834632
hosts_corrupt_100	0.9894327	+/- 0.0053556855
num_domains_5	1.2030568	+/- 0.10389547
num_domains_10	2.419214	+/- 0.14270231

Measures Considered

- We defined the *service by an application to be* *improper* if its replication group suffers a Byzantine fault: *a third or more* of the *currently active* replicas are corrupt.
- We defined several measures on the model to assess the survivability of the architecture
  - ***Unavailability*** for an *interval*: fraction of time the service was improper for an interval;
  - ***Unreliability*** for an *interval*: the probability that service was improper at least once during the interval;
  - ***Number of replicas*** of an application uncorrupt at a given time *instant;*
  - ***Fraction of corrupt hosts*** in a domain when the domain is excluded; and
  - ***Fraction of excluded domains*** at a given time *instant*.

Base Input Parameter Values

- Our main purpose was to demonstrate the efficacy of the approach and do a comparative analysis of various design choices by studying *trends*, and study system's sensitivity to important parameters.
- Several parameter values varied during studies, but unless otherwise specified, following values were used:
  - Cumulative base attack rate on the system was 3 successful intrusions per hour. Actual attack rate will generally be higher as a result of factors such as attack spread within domain and quick intrusions into replicas/manager on an intruded host.
  - Cumulative false alarm rate was 2 false alarms per hour.
  - Distribution of attacks on a host (OS and services): 80% script-based, 15% more exploratory, and 5% innovative.
  - Intrusion detection probabilities: 90% for script-based on hosts, 75% for more exploratory on hosts, 40% for innovative on hosts, 80% for replicas and managers.
  - Infiltration of a host increased the chances of corruption of other entities on the host five-fold.
  - A corrupt replica exhibited corrupt behavior twice per hour.

Input Parameter Values

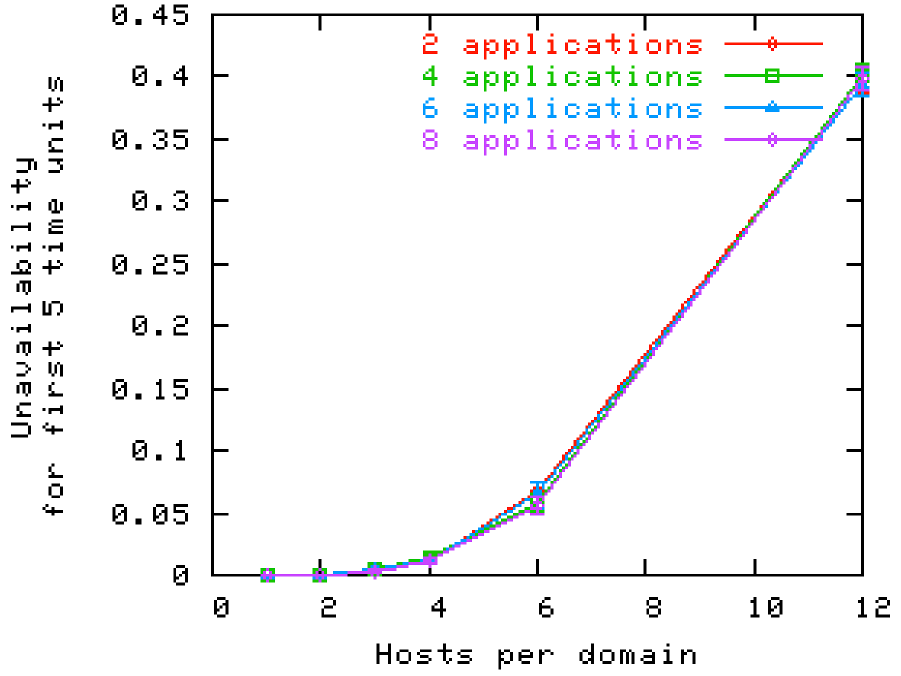
- The type and accuracy of input parameter values depends on the stage of development of the system being validated, and knowledge about possible attackers. Even if accurate input parameter values not available, the model can be used to:
  - Study the trends in the system's security, which can be used to *guide the system design process*, by making design choices that improve the survivability of the system.
  - Provide insights into requirements of particular system components to achieve desired overall survivability, e.g., maximum false alarm rate, required detection probability, maximum reaction and restoration time.
- Modeling-based validation can be used throughout the lifecycle of an intrusion-tolerant system, exploring the design space and making appropriate choices at each step of the design process.

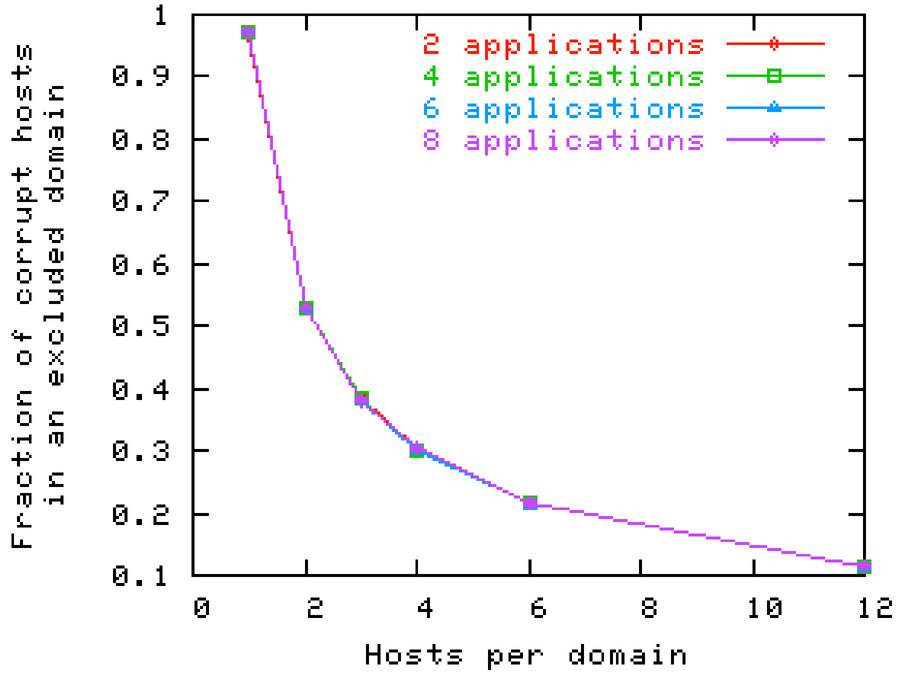
Result Overview

- We used the Möbius tool to design the SANs, define the measures on the model, and design the studies and obtain results via simulation.
- We conducted the following studies:
  - Determination of preferable distribution of hosts into domains by comparison of intrusion tolerance of the system for
    - different distributions of a constant number of hosts into domains
    - different numbers of hosts distributed into a fixed number of domains
  - Comparison of the relative efficacy of the host-exclusion and domain-exclusion management algorithms.

Impact of Different Distributions of a Constant Number of Hosts into Domains

- We distributed 12 hosts into 1,2,3,4,6 or 12 domains. We considered applications with 7 replicas each.

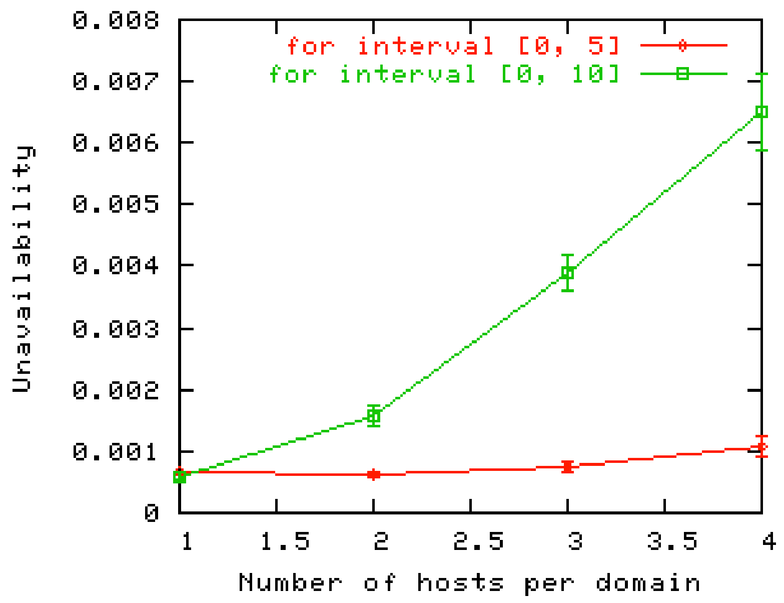


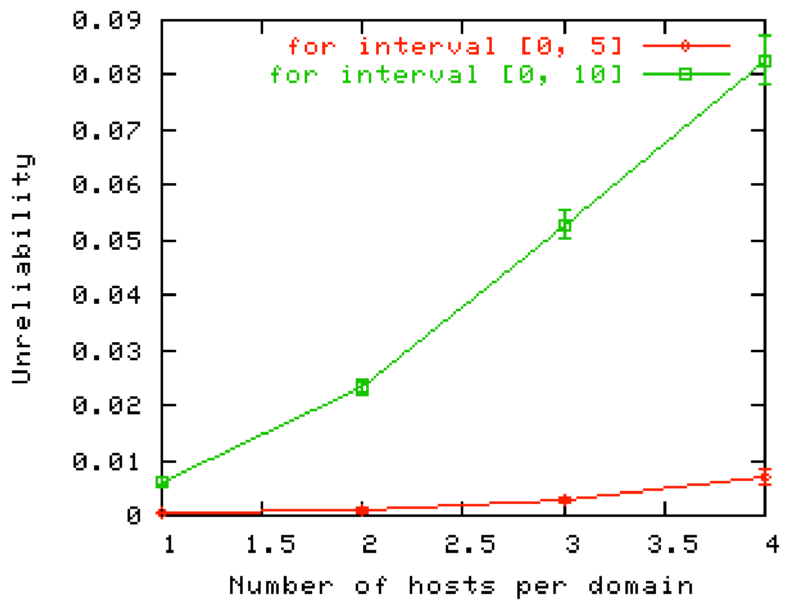


- System more available with fewer hosts per domain, because that allows for more domains and we do not run out of domains to restart replicas when several domains have been excluded.
- Resources are wasted when we have more hosts per domain, since corruption of a small fraction of hosts results in an exclusion of a large number of hosts.

Impact of Different Numbers of Hosts into a Constant Number of Domains

- Determine gains by putting more hosts per domain into a fixed number of domains, and evaluate the cost/benefit tradeoff.
- Number of domains was fixed at 1. Number of hosts per domain was varied from 1 to 4. The study had 4 applications with 7 replicas each.



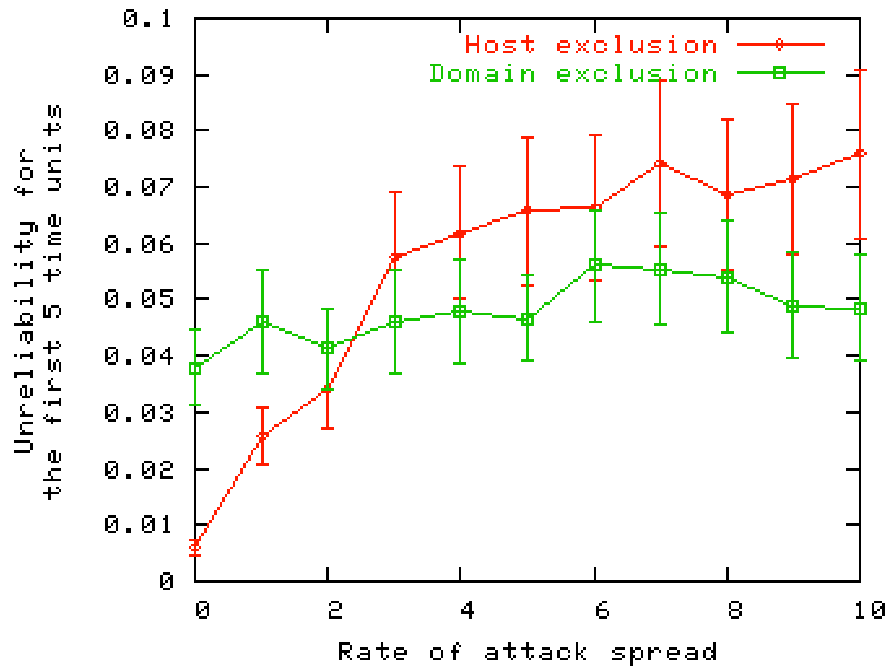


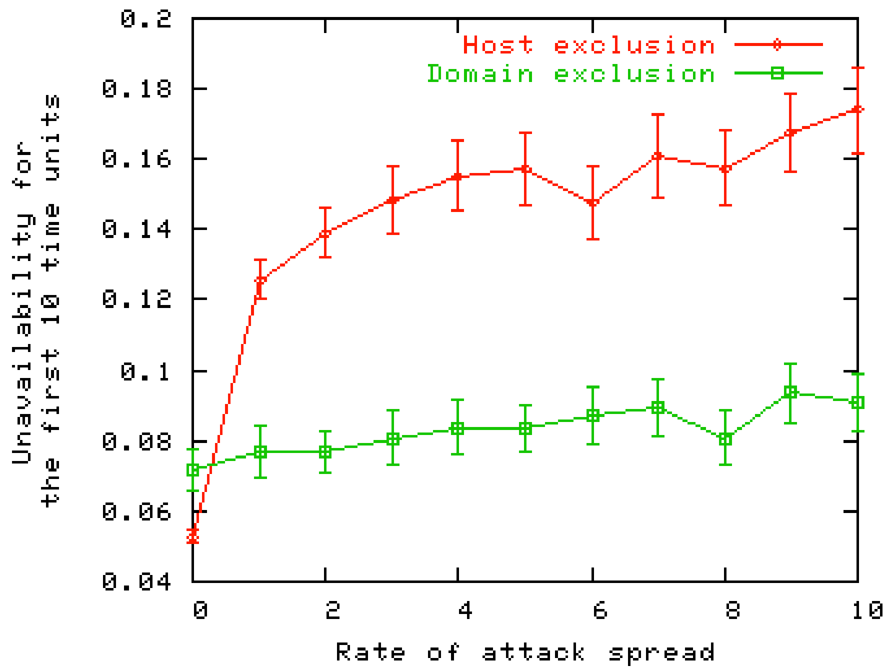
- System more available when fewer hosts per domain:
  - Probability of intrusion into a host same for all experiments
  - More hosts imply greater chance of *one* of them being corrupted, resulting in exclusion of its domain (upon detection).

Comparison of Domain-exclusion and Host-exclusion Management Algorithms

- Two ways to respond to detection of corruption of a host or a replica:
  - Exclude the entire domain that contains the host (preemptive response).
  - Exclude only the detected host, saving resources.
  - Presence of a corrupt host increases the rates of successful intrusions into other hosts in its domain.
  - Attack-spread is a model parameter characterizing how much influence the corruption of host has on attack rates for other hosts, and how quickly this influence spreads.
  - We studied the efficacy of the two management algorithms for different rates of attack spread.
  - We used the following parameter values:
    - 10 domains with 3 hosts per domain
    - 4 applications with 7 replicas each
    - Attack spread rate of varied from 0 (low) to 10 (high). A spread rate of 5 or more should be considered quite high.

Comparison of Domain-exclusion and Host-exclusion Management Algorithms





- In the short run (5 hours):
  - Host exclusion performs better for low attack spread rates, domain exclusion slightly outperforms for high attack spread rates.
- In the long run (10 hours):
  - Domain exclusion outperforms host exclusion for most values of attack rates.
- Domain exclusion scheme relatively insensitive to attack spread rate.

Summary

- Probabilistic evaluation/validation is an viable technique for choosing among design alternatives and validating intrusion-tolerant systems
- It should be used in all phases of a system's lifecycle: requirement specification, early/detailed design, implementation, testing, deployment, and maintenance
- Models are useful for making comparative studies and evaluating design alternatives, even if exact parameter values are not known
- Better parameter value estimation is necessary, for implemented systems, to quantify intrusion tolerance obtained
- Much more work is needed to build better models, particularly better attack models, and to better determine input parameter values!

Thank you!

for more information, see our papers at DSN03, and LADC03, found at

http://www.crhc.uiuc.edu/PERFORM/