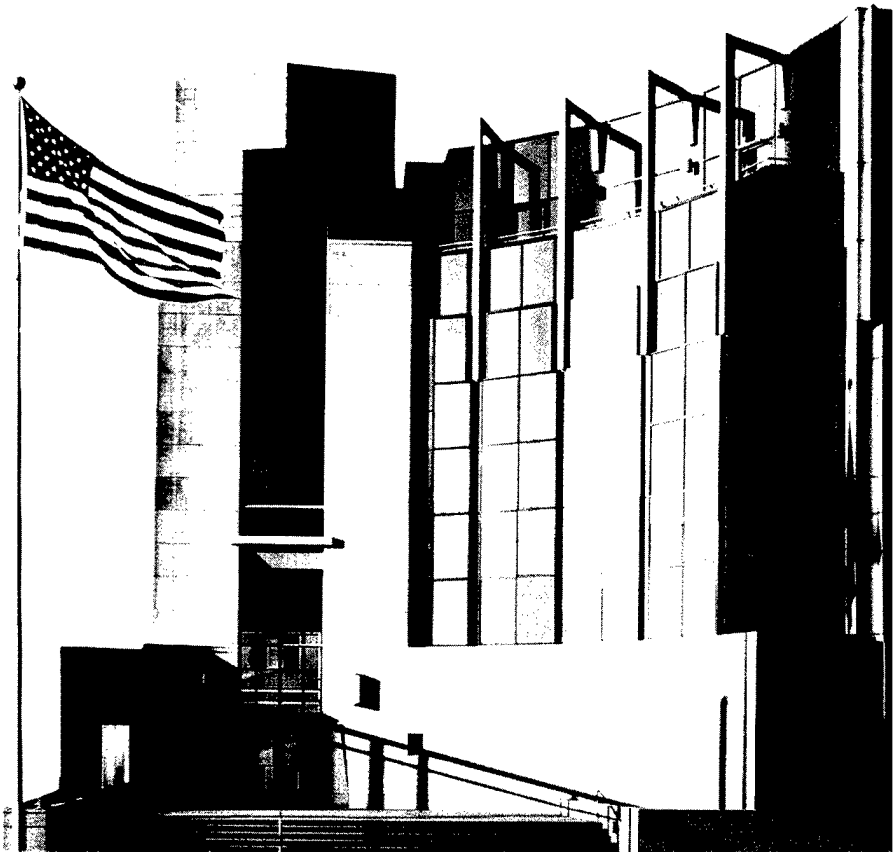
Carnegie Mellon
Software Engineering Institute

Foundations for Survivable System Development: Service Traces, Intrusion Traces, and Evaluation Models

Richard C. Linger
Andrew P. Moore

October 2001



TECHNICAL REPORT
CMU/SEI-2001-TR-029
ESC-TR-2001-029

Carnegie Mellon University does not discriminate and Carnegie Mellon University is required not to discriminate in admission, employment, or administration of its programs or activities on the basis of race, color, national origin, sex or handicap in violation of Title VI of the Civil Rights Act of 1964, Title IX of the Educational Amendments of 1972 and Section 504 of the Rehabilitation Act of 1973 or other federal, state, or local laws or executive orders.

In addition, Carnegie Mellon University does not discriminate in admission, employment or administration of its programs on the basis of religion, creed, ancestry, belief, age, veteran status, sexual orientation or in violation of federal, state, or local laws or executive orders. However, in the judgment of the Carnegie Mellon Human Relations Commission, the Department of Defense policy of "Don't ask, don't tell, don't pursue" excludes openly gay, lesbian and bisexual students from receiving ROTC scholarships or serving in the military. Nevertheless, all ROTC classes at Carnegie Mellon University are available to all students.

Inquiries concerning application of these statements should be directed to the Provost, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, telephone (412) 268-6684 or the Vice President for Enrollment, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, telephone (412) 268-2056.

Obtain general information about Carnegie Mellon University by calling (412) 268-2000.

Carnegie Mellon
Software Engineering Institute
Pittsburgh, PA 15213-3890

Foundations for Survivable System Development: Service Traces, Intrusion Traces, and Evaluation Models

CMU/SEI-2001-TR-029
ESC-TR-2001-029

Richard C. Linger
Andrew P. Moore

October 2001

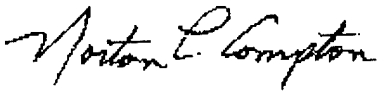
Survivable Systems

This report was prepared for the

SEI Joint Program Office
HQ ESC/DIB
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER



Norton L. Compton, Lt Col, USAF
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number F19628-00-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (http://www.sei.cmu.edu/publications/pubweb.html).

Table of Contents

List of Figures

Acknowledgements

The work described in this paper was partially funded by DARPA (Order #J884-01). Substantial contributions to the concepts were made by the members of the Survivable Systems Working Group: Gwen Walton, University of Central Florida; Ann Sobel, Miami University of Ohio; and Alan Hevner, University of South Florida. The authors gratefully acknowledge the suggestions provided by Tom Longstaff, Bob Ellison, Howard Lipson, and Nancy Mead.

Abstract

Survivability is a new branch of dependability. It addresses explicit requirements for restricted modes of operation that preserve mission-critical essential services in adverse operational environments. A survivable system is one that satisfies its survivability specification of essential services and adverse environments. On the system side, survivability specifications can be defined by *essential-service traces* that map *essential-service workflows,* derived from user requirements, into system component dependencies and required survivability attributes. On the environment side, survivability specifications can be defined by *intrusion traces* that map *intruder workflows,* derived from *attack patterns,* into compromisable system components. Survivability design applies resistance, recognition, and recovery strategies to maintain essential-service workflows where possible despite compromised components. Test environments for survivable system implementations can be defined by *survivability evaluation models* that merge essential-service and intruder workflows into usage-based, statistically valid test suites. This paper describes the initial results of research in these areas.

1 Survivable System Concepts

Modern society is irreversibly dependent on large-scale critical infrastructure systems to sustain quality of life, economic growth, and national security. As a result, society faces unquantified, but generally acknowledged as substantial, risks of system failure or compromise with potentially serious consequences. Sectors such as defense, telecommunications, energy, finance, and healthcare are potentially affected. Critical infrastructure systems share a dependency on large-scale computing and communication systems for operation and control. These systems exhibit powerful functionality for managing complex processes, extraordinary complexity that challenges intellectual control, extensive use of commercial off-the-shelf (COTS) components of uncertain reliability and quality, inflexible behavior that limits survivability, and cascade failure effects across interdependent *systems of systems*[1]. Market forces have long rewarded cost effectiveness over survivability in design and evolution of these systems.

As a new branch of dependability, the discipline of survivability addresses engineering methods for the analysis and design of systems that satisfy mission requirements for full services in benign environments and reduced but critical services in adverse environments. Survivable systems are intended to provide essential services despite the shock of adverse events. Essential services are mission-critical operations that must continue despite attacks, failures, or accidents [Ellison 99]. Our work is focused on attacks and intrusions, in the knowledge that the results achieved will be valuable in dealing with failures and accidents as well. Survivability requires capabilities to resist the effects of adverse environments, to recognize when these effects have occurred, and to recover from them in a timely manner.

Figure 1 depicts our dual-thread, three-stage approach to survivability research and development. We believe that survivability must be addressed from both the system side, in terms of specification and design for essential-service preservation, and from the environment side, in terms of intrusion specification and analysis. We further believe that rigorous foundations are a required basis for representation and reasoning in both areas, and for the definition of repeatable engineering practices. This paper describes initial work in the foundations stage for survivability specification and intrusion specification, as well as survivability evaluation models that draw upon both of these areas.

[1] A *system of systems* is a composite system composed of individual systems that may be tightly or loosely coupled. Such systems are prone to cascade effects of intrusions and compromises that propagate across system boundaries.

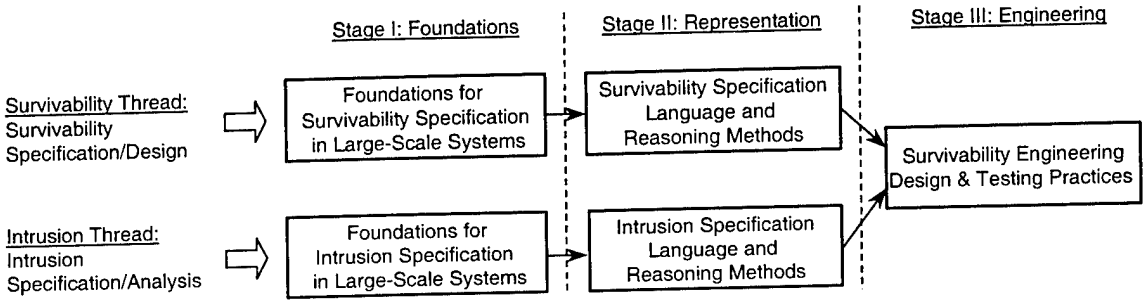


Figure 1: Dual-Thread, Three-Stage Approach to Survivability Research and Development

The requirements for essential services can be specified in terms of workflows, which can be refined into essential-service traces that define the sequencing of system architecture components and artifacts that are required to satisfy the workflows. These traces can be annotated with required survivability attributes. Intruder workflows can be derived from predefined attack patterns, and likewise refined into intrusion traces that define potentially compromisable system components and regions. Both essential-service and intruder workflows can inform the definition of evaluation models for survivability testing. Figure 2 depicts the relationship of survivability activities and work products to the overall development life cycle. Service traces, intrusion traces, and evaluation models are embedded within and support the larger activities of system specification, design, and testing. Survivability requirements and attack patterns drive the definition of essential-service and intruder workflows, respectively. These workflows are, in turn, expanded into architecture traces, which drive the selection and integration of survivability strategies in design and contribute to the definition of the testing environment.

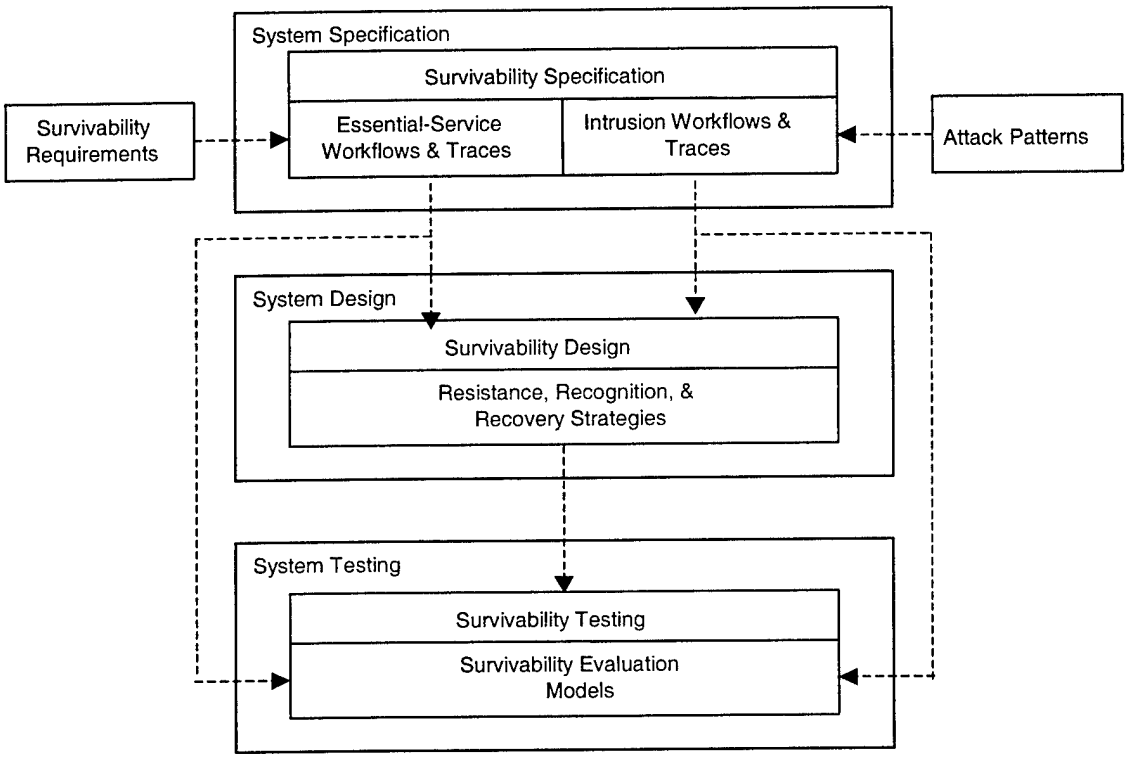


Figure 2: Survivability Relationship to Life-Cycle Activities

2 Service Traces for Survivability Specification

Current investigations in the foundations area of the survivability thread of Figure 1 involve essential-service workflows, corresponding service traces of essential-system components, and a computational approach to survivability attributes associated with traces. In today's world of large-scale, distributed systems of systems with indeterminate boundaries and complex asynchronous behavior, workflows and their architecture traces developed according to sound foundations provide rigorous and systematic representations for specification and design under intellectual control.

2.1 Essential-Service Workflows

Critical enterprise missions are carried out through the managed use of essential services provided by large-scale systems of systems composed of computation, communication, and human components. Services initiated by users at simple desktop machines may traverse extensive paths of hardware, software, and communications links, often astonishing in their complexity, to affect operations in a factory next door or a bank a continent away. Such paths may link many computing domains and communication media, and may require many transformations along the way.

Essential services can be defined by workflows [Hayes 00]. In simplest terms, a workflow is a directed graph composed of tasks, information elements, decisions, and flows. Workflows can define individual transactions or combinations of transactions. Tasks are nodes in the graph that define units of work carried out by humans or systems. Tasks accept input information and produce output information. Information elements are defined abstractly in terms of semantic content and concretely in terms of data. For example, a "get account balance" task could accept as input an "account number" and produce as output the corresponding "account balance." Decisions are predicates on information elements that determine the flow of control: for example, if "account balance < 0" then invoke "owner notification" task. Flow is defined by the arcs between nodes. Workflow nodes and arcs can be annotated with attributes that define required properties such as security and survivability. Large-scale systems typically provide many essential services that can be specified as workflows.

Because workflows contain decision points, they embody multiple paths from entry to exit, with every path representing a possible sequence of tasks and outcomes. In terms of control flow, both workflows and each of their contained tasks can be defined as single-entry, single-exit structures. Such structures exhibit important properties for refinement and decomposition, and permit workflows to be expressed at multiple levels of abstraction with full referential transparency. Any sequential workflow can be expressed in terms of composition, alternation, iteration, and concurrent structures.

2.2 Essential-Service Traces

An essential-service trace is a refinement of an essential-service workflow. It identifies the architecture components that every task and decision in an essential-service workflow requires for execution. A trace illuminates the hardware, software, communication, and human components that support the service and are thereby essential to its availability.

In a trace: workflow tasks are refined into uses of architecture components (both human and automated); information elements are refined into data; and decisions are refined into predicates on data values. For example, a "get account balance" task may require invocations of, say, "account manager," "database," "report generation," and "user interface" components. Such traces often reveal unforeseen and disturbing dependencies. For example, an essential-service workflow requiring the high availability of telecommunication services may specify the use of redundant carriers as a backup for potential outages. The trace refinement of the service could reveal, however, that the presumed-redundant carriers lease fiber-optic lines from the same provider, and that the service is in fact dependent on a single, fiber-optic line and its hardware and software controls—a clear single point of failure or compromise. Once identified, such dependencies can be redesigned to improve survivability attributes.

2.3 Relational Specification of Trace Components

It is invariably the case that particular system components (e.g., database or business-rule components) will experience many uses in many workflows. In the operational use of a system, workflows can be sequenced and interleaved by users in unpredictable ways. In fact, a principal design objective in large-scale systems is the coordination and synchronization of multiple uses of components specified by workflows. Furthermore, the components of large-scale, distributed systems respond asynchronously to a blizzard of inputs whose innumerable interleavings are essentially unknowable. Yet every asynchronous use of a component may change its current state and thereby its responses to future uses. Thus, any operational use of a workflow component can encounter any possible state at all in execution and receive a response determined by that state. Because of this, when service traces are specified, the behavior of every component must be defined in terms of all possible responses resulting from all possible component states that may be encountered. Such a relational specification defines a set of all the responses accumulated from all the uses of a component in all the traces

within which it appears. In this way, completeness and consistency can be achieved in trace definitions. It may seem a formidable undertaking to enumerate all the possible states of a component, but this is not necessarily the case. It is usually sufficient to define equivalence classes across potential states that dramatically reduce specification complexity. For example, in accessing records from a database component, responses can be partitioned into "present" and "not present," with no immediate need for elaboration of the multiple circumstances leading to a "not present" response.

Incorporating the relational specifications of component behavior means that every use of a component in the workflows reflects all possible outcomes for that use, no matter what history of asynchronous use it may have experienced up to that point. This completeness and consistency of specification permits the paths of a trace to be treated as sequences of tasks and decisions with full functional and compositional properties for simplified reasoning and analysis, rather than as views of asynchronous processes with indeterminate outcomes, which are difficult to reason about. Relational component specifications may reveal additional decision points unanticipated in the original workflows, which can be updated to reflect the expanded set of possible paths and outcomes.

2.4 Computational Survivability

Substantial effort has been devoted to developing descriptive and largely subjective characterizations of survivability as a non-functional system property. While useful methods have emerged from this work, much remains to be done. Rather than focusing on non-functional descriptions, we consider an alternative approach and ask how survivability can be defined, computed, and acted upon as a dynamic characteristic of system operation. That is, we wish to define computational survivability as a function to be computed, rather than as a subjective description of a property to be achieved. While such a function relies on what can be computed and may differ thereby from traditional, non-functional views, it may permit new approaches to survivability analysis, design, and operation. In illustration, a function implementing computational survivability, centralized or decentralized within the control structure of a system, could accept as stimuli the status of system services and compromises, and produce as responses modified traces that maintain the survivability properties of essential workflows where possible. We believe that computational survivability is a fruitful area for research and development.

Survivability requirements can be associated with system-component uses embedded within workflow traces. For example, a trace for an essential service may employ a database component in its sequencing of component uses. In this usage, the database must satisfy a high level of specified survivability. However, other uses of the same database in workflows for non-essential services would carry less stringent specifications of survivability requirements. Workflow traces provide a semantic framework for the specification, analysis, and composition of survivability properties.

As noted above, a primary control task in large-scale systems is managing the composition of system components to satisfy workflow trace specifications. System-control functions reconcile trace specifications with available system components, and are a natural vehicle for implementing survivability management based on the dynamic network and component capabilities and availabilities [Sullivan 99]. The concept of a service-flow machine (SFM), in analogy to a data-flow machine, either centralized or decentralized within the architecture of a system, can embody trace-management functions that include dynamic survivability management through a variety of strategies, including alternate communication paths, resource substitutions, state purging, alternate provisioning, and system reinitialization and reconfiguration [Hevner 02]. An SFM abstraction could be designed and instantiated in a variety of forms and technologies, depending on the survivability requirements, network configuration, and operational environment.

3 Intrusion Traces for Adverse Environment Specification

Current investigations in the foundations stage of the intrusion thread of Figure 1 involve intruder workflows derived from attack patterns and corresponding intrusion traces of compromisable system components. We define an intruder workflow and its encompassing attack tree as

- intruder workflow – the sequence of steps that an intruder executes to compromise the survivability of an enterprise
- attack tree – a set of hierarchically organized intruder workflows that result in a common survivability compromise

Enterprise-specific intruder workflows and attack trees are built from reusable attack patterns:

- attack pattern – a generic representation of a deliberate, malicious attack that commonly occurs in specific contexts
- attack profile – a set of related attack patterns defined in terms of a common architectural reference model

Finally, intrusion traces help to identify countermeasures to resist, recognize, and recover from intrusions. Intrusion traces map the intruder workflow onto the enterprise architecture.

We distinguish between an intrusion and an attack as follows. An attack is malicious activity that may or may not, by itself, compromise an enterprise's survivability. An attack may or may not successfully accomplish an attacker's objectives. Even if it does, an individual attack may only increase an attacker's ability to compromise survivability, without actually causing the compromise. An intrusion, on the other hand, is a sequence of successful attacks that results in a compromise to enterprise survivability.[2] Attackers become intruders once they execute such intrusions.

This section introduces the structures supporting these concepts in more detail. We describe how attack patterns can be used to build intruder workflows that are organized into attack trees. Intrusion traces map these workflows onto the enterprise architecture to help identify

[2] This definition of intrusion suggests that attack trees may be more appropriately called intrusion trees. We, however, use the more common phrase *attack trees* to avoid confusion [Schneier 00].

countermeasures to the intrusions. Our documentation approach provides a basis for using attack data to improve the design and analysis of secure and survivable information systems. We expect that security analysts will be able to use the structures described to document ongoing attacks derived from real attack data or hypothesized attacks from projected trends of intruder behavior. Further, we expect that information system designers and analysts will be able to use the attack patterns to make systems more secure and survivable.

3.1 Intruder Workflows Organized Into Attack Trees

The large number of intruder workflows relevant to any nontrivial enterprise necessitates a scheme to organize related workflows. Attack trees provide such an organizational scheme [Salter 98, Schneier 99, Schneier 00]. They refine information about attacks by identifying the compromise of enterprise security or survivability as the root of the tree. The ways that an attacker can cause this compromise are refined incrementally as lower level nodes of the tree. For example, suppose *Mal* is a malicious competitor to a business, called *Biz*, that sells some product. Mal may compromise Biz's ability to make a profit by

1. hampering Biz's development of the product
2. disrupting Biz's sales of the product
3. undermining customer demand for Biz's product

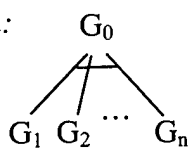
Each of these attack classes can be refined as a separate branch of the attack tree.

An enterprise typically has a set, or forest, of attack trees that are relevant to its operation. The root of each tree in a forest represents an event that could significantly harm the enterprise's mission. Each attack tree enumerates and elaborates the ways that an attacker could cause the event to occur. Each path through an attack tree represents a unique intrusion on the enterprise. We decompose a node of an attack tree as one of the following:

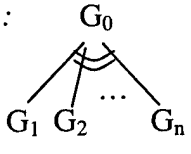
- a set of attack subgoals that is represented as an AND decomposition. All of these goals must be achieved for the attack to succeed.
- a set of attack subgoals that is represented as an OR decomposition. If any of these goals is achieved, the attack succeeds.

We represent decompositions graphically as follows:

AND-decomposition:

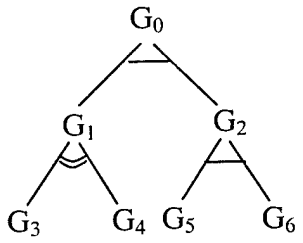


OR-decomposition:



The AND-decomposition represents a goal G_0 that can be achieved if the attacker achieves all of the goals G_1 through G_n. The OR-decomposition represents a goal G_0 that can be achieved if the attacker achieves any one of goals G_1 through G_n. In practice, we often represent attack trees textually, since the graphical representation can be awkward for nontrivial attack trees.

Attack trees consist of any combination of AND- and OR-decompositions. We generate individual intruder workflows from an attack tree by traversing the tree in a depth-first manner. For example,



generates the intruder workflows

$\langle G_3, G_5, G_6 \rangle$
$\langle G_4, G_5, G_6 \rangle$

In general, leaf goals are added onto the end of intruder workflows as they are generated. OR-decompositions cause new workflows to be generated. AND-decompositions cause existing workflows to be extended. Intermediate nodes of the attack tree do not appear in the intruder workflows, since they are elaborated by lower level goals.

Attack trees allow the refinement of attacks to a level of detail chosen by the developer. They exhibit the property of referential transparency as characterized by Prowell:

> *"Referential transparency implies that the relevant lower level details of an entity are abstracted rather than omitted in a particular system of higher level description, so that the higher level description contains everything needed to understand the entity when placed in a larger context" [Prowell 99].*

This property permits the developer to explore certain attack paths in more depth than others, while still allowing the developer to generate intruder workflows that make sense. In addition, refining the branches of the attack tree generates new leaves resulting in intruder workflows at the new lower level of abstraction.

3.2 Attack Patterns Organized Into Attack Profiles

The practicality of attack trees to characterize attacks on real-world systems depends on being able to reuse previously developed patterns of attack. We describe two structures that support such reuse: an attack pattern for characterizing an individual type of attack, and an attack profile for organizing attack patterns to make it easier to search for and apply them.

An attack pattern contains the overall goal of the attack specified by the pattern, a list of preconditions for its use, the steps for carrying out the attack, and a list of postconditions that are true if the attack is successful. The preconditions include assumptions that we make about the attacker or the state of the enterprise that are necessary for an attack to succeed. Example preconditions include the skills, resources, access, or knowledge that attackers must possess, and the level of risk that they must be willing to tolerate. The postconditions include knowledge gained by the attacker and changes to the enterprise state that result from successfully carrying out the attack steps when the preconditions hold.

We further organize related attack patterns into an encompassing attack profile. Attack profiles contain a common reference model, a set of variants, a set of attack patterns, and a glossary of defined terms and phrases. The reference model represents an architecture template with parameters that may include the variants. The attack patterns are also defined in terms of the variants. As we will describe more fully in the next section, attack profiles are specified independently of any particular enterprise. An enterprise whose architecture is consistent with a profile's reference model may use the profile's attack patterns, once instantiated, to help construct the attack trees that are relevant to the enterprise's operation. Different attack profiles may address different levels of attacker access, resources, and skills, as well as different configurations of system components. Therefore, different attack profiles may help refine an enterprise-specific attack tree along different lines of attack.

As in the example in Section 3.1, Mal may want to disrupt the sales of Biz by attacking Biz's Web site. A SYN Flood attack pattern can severely degrade the performance of a Web site by repeatedly transmitting TCP SYN packets to the Web server from a forged, bogus origin. The resulting flood of interminable connection requests overwhelms the Web site so that it cannot process requests by legitimate customers. A precondition of this attack is that Mal knows the Internet Protocol (IP) address of Biz's Web site. Such an attack pattern may be part of an attack profile in which the reference architecture allows Internet-based attacks on an enclave protected at its boundary by a firewall. Variants of this attack profile include the component of the enclave under attack and the type of firewall at the enclave boundary.

3.3 Refining Attack Trees Using Attack Patterns

As shown in the flowchart in Figure 3, an attack tree can be refined from the root-node compromise as a combination of manual extensions and pattern applications. Manual extensions depend greatly on the security expertise of the person developing the attack tree. Pattern application also depends on such expertise, but to a lesser extent. Some of this security expertise is built into an attack-pattern library.

A good attack-pattern library provides a set of attack profiles that are rich enough to characterize the attacks that may take place on a broad range of enterprise architectures. Refining a particular enterprise's attack tree involves first finding those attack profiles that are consistent with the enterprise architecture. The developer searches the attack patterns of consistent attack profiles for a refinement of an attack path contained in the enterprise attack tree. Once found, the developer can appropriately instantiate and apply the attack pattern to extend the enterprise attack tree. This process of pattern application intermixed with manual extension continues until the attack tree is refined sufficiently. The decision of when to halt the process is at the discretion of the developer.

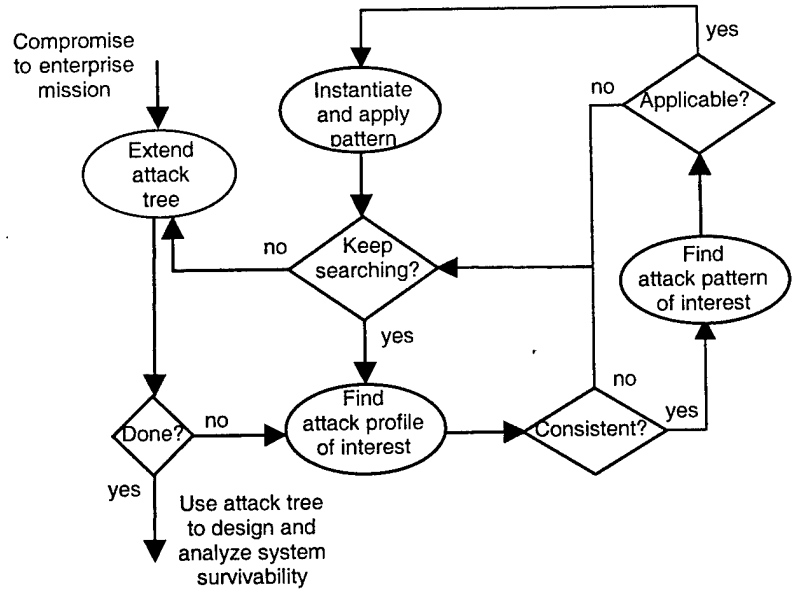


Figure 3: Attack-Tree Refinement Process

As mentioned previously, the reference model associated with an attack profile can be viewed as an architecture template. The parameters of this template are the reference model variants. If a set of values for these variants exists that unifies the attack profile's reference model with some portion of the enterprise architecture, we say that the attack profile is *consistent with* the enterprise architecture. The attack patterns associated with the profile are written with respect to the profile's reference model and in terms of the profile's variants. These attack patterns are, therefore, relevant to the enterprise architecture.

Determining which attack profiles are consistent with the enterprise architecture is only the first step. Analysts must also determine which attack patterns in consistent profiles help refine the enterprise attack tree. This requires identifying a pattern whose goal helps to achieve the goal identified at an attack-tree node. We say that such patterns, when properly instantiated, are *applicable to* the enterprise attack tree. In our previous example, the SYN Flood attack pattern is applicable to the second branch of Biz's attack tree (disrupting sales of the product) where the component under attack is Biz's Web server.

3.4 Analyzing Attack Trees Using Intrusion Traces

Intrusion traces map each intrusion path through an attack tree onto the enterprise architecture. This helps identify those components that can be compromised from the intrusion and the impact to the enterprise mission that results. The enterprise architecture can then be analyzed by asking resistance or recovery questions at each of the attack tree nodes. Resistance questions ask, "How can we prevent an attacker from successfully traversing this node to achieve the attacker's goal?" Of course, the answer to such questions may not always be a cost-effective or practical solution. Fundamental to the goal of survivability is recognizing when an attack that we cannot effectively resist takes place and executing recovery plans. We thus ask "How can we detect an attacker during an attempted attack or after a successful attack?" and "How can we react to this detection?"

In our running example, an intrusion trace of the SYN Flood attack would clearly identify Biz's Web server as a compromisable component. Biz may decide to do one of the following:

- Reduce the effectiveness SYN Flood attacks generally by decreasing the amount of time a TCP connection request takes to time-out.
- Detect the attack when it occurs and free up additional space so that customer requests can be handled.

Attack trees provide a powerful mechanism to document the multitude of diverse types of attacks, to abstract from intrusion details as a buffer against attack volatility, and to suggest improvements to requirements and design. They are, however, only a relatively small part of the answer as to how to use intruder workflows and intrusion traces to design more survivable systems. The lack of accurate adversary models and risk-analysis methods obstructs the

progress on this front. A rich attack-pattern library populated with attack patterns at the right level of abstraction is needed to build enterprise attack trees more systematically. Finally, the lack of robust resistance, recognition, and recovery countermeasures hampers our ability to construct survivable systems. Overcoming these obstacles will require a truly interdisciplinary effort.

4 Survivability Evaluation Models

It is well known that the set of possible executions of a large-scale system forms an essentially infinite population. All testing is in effect sampling from that infinite population. No testing process, no matter how well funded and conceived, can execute more than a minute portion of all the possible executions. Thus, the real question is how to draw the finite sample so as to maximize the value of the testing process. If the sample is randomized based on projected usage, the results of the testing process can be used to estimate system performance on all the test cases in the infinite population that could not be executed. Such statistical usage-based testing produces scientific measures of system quality, just as has been done in hardware engineering for years.

When a population is too large to permit exhaustive analysis, as is indeed the case in large-scale information systems, a statistically correct sample must be defined as a basis for making valid inferences about the population as a whole [Prowell 99]. In such a statistical testing protocol, the environment of usage is modeled in terms of states of use and associated probabilities of transitions among states. Usage models can be represented conveniently in formal grammars or Markov chains. Every possible usage scenario is represented in the model and potentially generated as a test case by traversing the model according to its transition probabilities. Transition probabilities among states can be determined from historical or projected usage data for a system. Where extensive field usage data exists, probabilities may be known in detail. For new systems, probabilities may be estimated initially and refined as usage data accumulates. When complete information is not available, usage models can be represented as a system of constraints, and transition probabilities can be generated through mathematical programming as the solution to an optimization problem.

Survivability workflows and their essential-service traces enumerate steps in system usage that can define, in conjunction with all other system usage, the structural characteristics of usage models for statistical testing. Intrusion workflows and their traces likewise contribute to the definition of usage-model structures. For survivability evaluation, a usage model for a system must embody essential and non-essential service workflows, as well as intrusion workflows, all of which are interwoven according to usage probabilities to represent the spectrum of expected system usage by both legitimate users and attackers. Such models support generations of test suites whose execution results permit the estimation of system performance for the defined usage.

While usage-based statistical testing has proven to be a powerful tool for system quality and reliability assessment, it is an open question whether the benefits of the approach can be achieved in survivability testing against intrusions. Two major issues must be addressed: what methods can be used to define intrusion probabilities, and how unforeseen intrusion strategies not represented in the usage models can be accommodated in the analysis [McHugh 00]. The high potential value of statistical testing for survivability analysis motivates our interest in developing solutions to these problems.

5 Future Work

We intend to continue elaborating on the concepts described in this paper through

1. defining the mathematical properties of essential-service and intrusion traces and evaluation models
2. developing language representations for traces and evaluation models
3. defining engineering practices for trace and evaluation-model specification, analysis, and design

References/Bibliography

[Anderson 94] Anderson, R. "Why Cryptosystems Fail," 47-61. *Proceedings of 12th Worldwide Congress on Computer and Communications Security and Protection.* Paris, France, June 1-2, 1994. Paris, France: M.C.I.- Manifestations et Commun. Int., 1994.

[Arbaugh 00] Arbaugh, W.A.; Fithen, W.L.; & McHugh, J. "Windows of Vulnerability: A Case Study Analysis." *IEEE Computer 33*, 12 (December 2000): 52-59.

[Ellison 99] Ellison, R.; Fisher, D.; Linger, R.; Lipson, F.; Longstaff, T.; & Mead, N. *Survivable Network Systems: An Emerging Discipline* (CMU/SEI-97-TR-013, ADA341963). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1997, revised 1999. Available WWW: <URL: http://www.sei.cmu.edu/publications/documents/97.reports/97tr013/97tr013abstract.html> (1999).

[Hayes 00] Hayes, J., et al. "Workflow Interoperability Standards for the Internet." *IEEE Internet Computing 4,* 3 (May/June 2000): 37-45.

[Hevner 02] Hevner, A.R.; Linger, R.C.; Sobel, A.E.; & Walton, G. "Flow-Service-Quality Structures: A Unified Engineering Framework for Survivable Distributed Systems. *Proceedings of the Hawaii International Conference on System Sciences.* Kona, Hawaii, January 7-10, 2002. Los Alamitos, CA: IEEE Computer Society Press, 2002.

[Howard 98] Howard, J. & Longstaff, T. *A Common Language for Computer Security Incidents* (SANDIA Report SAND98-8867). Livermore, CA: Sandia National Laboratories, October 1998. Available WWW: <URL: http://www.cert.org/research/papers.html> (1998).

[McHugh 00] McHugh, J. "The 1998 Lincoln Lab IDS Evaluation: A Critique," 145-161. *RAID 2000, Lecture Notes in Computer Science # 1907.* Berlin, Germany: Springer-Verlag, 2000.

[Mead 00] Mead, N.; Ellison, R.; Linger, R.; Longstaff, T.; & McHugh, J. *Survivable Network Analysis Method* (CMU/SEI-2000-TR-013, ADA383771). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, September 2000. Available WWW: <URL: http://www.sei.cmu.edu/publications/documents/00.reports/00tr013.html> (2000).

[Prowell 99] Prowell, S.; Trammell, C.; Linger, R.; & Poore, J. *Cleanroom Software Engineering: Technology and Process.* Reading, MA: Addison-Wesley-Longman, Inc., 1999.

[Salter 98] Salter, C.; Saydjari, O.; Schneier, B.; & Walner, J. "Toward a Secure System Engineering Methodology," 2-10. *Proceedings of the New Security Paradigms Workshop.* Charlottesville, Virginia, September 22-26, 1998. New York, NY: Association for Computing Machinery, 1999.

[Schneier 99] Schneier, B. "Attack Trees: Modeling Security Threats." *Dr. Dobb's Journal 24,* 12 (December 1999): 21-29.

[Schneier 00] Schneier, B. *Secrets and Lies: Digital Security in a Networked World.* New York, NY: John Wiley & Sons, August 2000.

[Sullivan 99] Sullivan, K.; Knight, J.; Du, X.; & Geist, S. "Information Survivability Control Systems," 184-192. *Proceedings of the 21st International Conference on Software Engineering.* Los Angeles, California, May 16-22, 1999. New York, NY: Association for Computing Machinery, 1999.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	October 2001	Final

4. TITLE AND SUBTITLE
Foundations for Survivable System Development: Service Traces, Intrusion Traces, and Evaluation Models

5. FUNDING NUMBERS
F19628-00-C-0003

6. AUTHOR(S)
Richard C. Linger and Andrew P. Moore

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

8. PERFORMING ORGANIZATION REPORT NUMBER
CMU/SEI-2001-TR-029

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
HQ ESC/XPK
5 Eglin Street
Hanscom AFB, MA 01731-2116

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
ESC-TR-2001-029

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	Unlimited

13. ABSTRACT (MAXIMUM 200 WORDS)

Survivability is a new branch of dependability. It addresses explicit requirements for restricted modes of operation that preserve mission-critical essential services in adverse operational environments. A survivable system is one that satisfies its survivability specification of essential services and adverse environments. On the system side, survivability specifications can be defined by *essential-service traces* that map *essential-service workflows,* derived from user requirements, into system component dependencies and required survivability attributes. On the environment side, survivability specifications can be defined by *intrusion traces* that map *intruder workflows,* derived from *attack patterns,* into compromisable system components. Survivability design applies resistance, recognition, and recovery strategies to maintain essential-service workflows where possible despite compromised components. Test environments for survivable system implementations can be defined by *survivability evaluation models* that merge essential-service and intruder workflows into usage-based, statistically valid test suites. This paper describes initial results of research in these areas.

14. SUBJECT TERMS	15. NUMBER OF PAGES
system survivability, information system security, Internet security, attack patterns, risk analysis, essential services, user workflow intrusions, system mission	34

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL