Security and Survivability Reasoning Frameworks and Architectural Design Tactics

Robert J. Ellison
Andrew P. Moore
Len Bass
Mark Klein
Felix Bachmann

September 2004

Network Systems Survivability
Software Architecture Technology

Technical Note
CMU/SEI-2004-TN-022

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).

Contents

List of Figures

List of Tables

Abstract

The Software Engineering Institute (SEI) has been investigating disciplined software architecture design for several years. The SEI approach includes a collection of "quality attribute reasoning frameworks" that understand both quality attribute reasoning and how architects design for the quality attribute under particular situations. The approach was first applied to the quality attributes of modifiability and performance. This report is an initial attempt to use the same method for the related quality attributes of security and survivability. The report includes an initial organization of security within the framework, a partial explication of elements of that framework, and three representative examples of existing security reasoning frameworks.

1 Introduction

The Software Engineering Institute (SEI) has been investigating disciplined software architecture design for several years. We have reported this investigation as snapshots of our current thinking in a series of reports [Bachmann 02, Bachmann 03a, Bachmann 03b]. These reports have dealt with the quality attributes of modifiability and performance. This report is an initial attempt to use the same approach for the related quality attributes of security survivability. We provide an initial organization of security within our framework and a partial explication of elements of that framework.

Our approach includes a collection of "quality attribute reasoning frameworks" that understand both quality attribute reasoning and how architects design for the quality attribute under particular situations. Another element of our framework is the concept of *general scenario* [Bass 01]. General scenarios provide a structured means of stating quality attribute requirements. System-specific quality attribute requirements—called *concrete scenarios*—are instances of general scenarios. In designing an architecture we start with a concrete quality attribute scenario and then determine the general scenario of which it is an instance. Concrete scenarios are requirements that must be satisfied and the corresponding general scenarios act as triggers to determine which quality attribute reasoning framework(s) will react in response to this requirement.

We begin by discussing the characteristics of a reasoning framework, the concepts they embody, and how they drive the design framework we are developing.

Our goal is to be able to predictably determine how an architecture responds to security threats and be able to improve that response. Achieving this goal requires

- determining what is meant by a security threat
- determining what is meant by a response to a security threat
- determining how to improve an architecture so that it has a better response to a security threat
- determining how to predict the response of an architecture to a security threat

1.1 Characterizing Threats and Responses

We characterize threats and responses in terms of general and concrete scenarios. Every threat is an occurrence of some sort. That occurrence can be premeditated, such as a distributed denial-of-service (DDoS) attack, or it can be accidental, such as an employee

inadvertently attempting to gain access to sensitive information. Furthermore, there is some response desired from the system when this occurrence happens. The response may be to continue service (in the event of a DDoS attack) or it may be to prevent or record access to the sensitive information.

In previous work we have characterized descriptions of quality attributes through general scenarios [Bass 03]. A general scenario is a system-independent description of a quality attribute requirement. It has six parts:

- a stimulus: a condition that needs to be considered when it arrives at a system
- a source of the stimulus: the entity (e.g., a human or computer system) that generated the stimulus
- an environment: the conditions under which the stimulus occurs; for example, when the system is in an overload condition
- the artifact affected by the stimulus
- a response: the activity undertaken after the arrival of the stimulus
- a response measure: the attribute-specific constraint that must be satisfied by the response

In the DDoS example, the stimulus is the attack itself. The source of the stimulus is from multiple distributed computer systems external to the system being attacked. The environment is, presumably, under normal operations. The artifact is the system being attacked, the response is that the system will continue normal operations, and the response measure is that normal response times are maintained.

A concrete scenario is a system-dependent instantiation of a general scenario. In the DDoS example, the instance would describe how many messages were expected in the attack and would make specific the response time requirement.

1.2 Improving an Architecture with Respect to a Security Threat

Architects have techniques that they apply to an architecture in order to improve the behavior of the architecture with respect to classes of security threats. Adding a firewall, for example, improves the architecture of the system with respect to accessing sensitive information. A firewall improves the architecture with respect to other conditions as well, but for the purposes of example we are only concerned with accessing sensitive information. A *software architectural tactic* is a transformation of a software or system architecture that improves its response with respect to a particular measure. A key element of this definition is that a tactic is a transformation of a software or system architecture. Using a physical key to restrict access to the room where the sensitive information is stored is a tactic for preventing unauthorized access to sensitive information, but it does not affect the software or system

architecture. We will elaborate software architectural tactics relevant to security later in this report, but two common examples are inserting a firewall and encrypting data.

Tactics are responses to classes of scenarios. For example, encrypting data does not help respond to a DDoS attack. Adding a redundant component does not help respond to an employee accidentally attempting to access sensitive information. Thus, it makes sense to couple tactics with the classes of threats to which they respond. Again, we will elaborate this idea later in this report.

1.3 Predicting the Response of an Architecture to a Threat

An ideal situation is to be able to predict the response of an architecture to a particular threat. Comparing the response of an architecture to a threat with the response of that architecture transformed by the addition of a tactic will enable a comparison of the benefits of using that tactic against the cost of implementing it.

The goal is to define a function that will predict the response measure for a particular architecture, given the stimulus from a concrete scenario. This requires that the architecture and the stimulus can be interpreted in terms that provide parameters for the function. We call such a function a *quality attribute model*. Furthermore, there must be a method for evaluating the function. We call such a method an *evaluation.*

Security has a shortage of such prediction functions, so we use performance as our example. Rate Monotonic Analysis (RMA) is a theory of performance that can be used to predict latency for periodic functions under certain scheduling disciplines. The response measure predicted from an RMA quality attribute model is latency. The parameters for this model are the period at which events arrive, the computation time for the tasks in the architecture, the scheduling discipline and priorities of the tasks, the number of processors involved, and some structural information about the relationship among the tasks.

Observe that not all of performance is covered by RMA theory. It does not cover cases where arrivals are stochastic rather than periodic, and there are other restrictions that determine whether it is relevant to a particular problem.

Our goal with respect to security is to have a collection of theories that can be used to predict the response of an architecture to a collection of stimuli of different types. The architecture can then use the predictions of these theories to assist in the process of architecture design.

1.4 Quality Attribute Reasoning Frameworks

Quality attribute reasoning frameworks are at the heart of the software architecture design framework that we are developing. They are also at the heart of the Predictability by

Construction theory that the SEI's Predictable Assembly from Certifiable Components (PACC) initiative is developing [Wallnau 03]. What we present here is a software architecture design perspective on reasoning frameworks that is consistent with but complementary to the PACC use of the term. In our context, reasoning frameworks provide the connection between architectural design decisions and quality attribute measures. A reasoning framework consists of the following portions:

1. **Scenario triggers.** A characterization of the set of general scenarios for which this reasoning framework is relevant. As we just said, we must be precise about the situations for which a particular theory applies. We use a set of general scenarios to characterize these situations.
2. **A set of quality attribute models.** Each quality attribute reasoning framework has a collection of quality attribute models characterized by an appropriate set of elements, relations, and properties.
3. **An evaluation.** A method for evaluating a quality attribute model.
4. **An interpretation.** The interpretation converts an architectural description into a set of parameters for a quality attribute model.
5. **A set of architectural tactics.** The architectural tactics that are used to improve the response of the architecture to the stimuli included in the scenario triggers.

With these elements, the outlines of a design process that satisfies a single quality attribute requirement can be seen. Begin with an architecture and the requirement expressed as a concrete scenario. Map the concrete scenario into the corresponding general scenario. Now choose a reasoning framework that has that general scenario as one of its triggers. A set of architectural tactics comes along with that reasoning framework. Determine the quality attribute model for the architecture. Transform the architecture using architectural tactics until the quality attribute model for the resulting architecture satisfies the concrete scenario. Of course, the design process is not as mechanical as this might sound. However, this general approach provides a conceptual basis for a methodical design process.

Extension of this process to multiple quality attribute reasoning frameworks and elaboration of the process is outside of the scope of this report. We are concerned here with identifying and discussing security reasoning frameworks.

2 Elements of Security and Survivability Reasoning Frameworks

This section discusses the primary characteristics of security and survivability reasoning frameworks in terms of the structure for quality attribute reasoning frameworks introduced above. We first describe the characteristics of security and survivability general scenarios that imply a range of triggers for security and survivability reasoning frameworks. We next describe security and survivability evaluators, from general evaluators in the area of security risk analysis to scenario-specific evaluators that address particular attack patterns. Finally, we start characterizing the set of architectural tactics used to achieve security and survivability. By collecting these tactics into groups based on the general scenarios to which they respond, we identify an initial collection of reasoning frameworks. At this point, we do not have the other portions of these reasoning frameworks (the evaluator, the set of quality attribute models, and the interpreter) but we have identified the scope that such reasoning frameworks would cover. Section 4 describes examples of these frameworks in more detail.

2.1 Security and Survivability General Scenarios

At a high level, a security or survivability general scenario is typically spurred by *attacks perpetrated or vulnerabilities exploited* and results in *resistance of the attack with no disruption of essential services or recognition and recovery of the attack so as to continue essential services despite the attack*. More specifically, the parts of a security and survivability general scenario are as follows:

A stimulus (a condition that needs to be considered when it arrives at a system): The stimulus is generally going to be a class of attack that leads to vulnerability exploitation. The triggering scenarios of a particular security and survivability reasoning framework are likely to share common properties of the attacker (e.g., the level of access) or common properties of the stimulated artifact (e.g., a specific architectural vulnerability).

A source of stimulus (the entity that generated the stimulus)*:* The source of the stimulus is generally going to be the attacker. Characterizing specific types of attackers is beyond the scope of this report. There is a plethora of books that describe the attributes and techniques of fairly unsophisticated but malicious individuals often called hackers or crackers. The characterization of more sophisticated attackers, such as industrial spies and international cyber-terrorists, is usually sensitive and sometimes classified.

An environment *(the conditions under which the stimulus occurs):* The environment is typically going to dictate the types of attackers that may be interested or able to cause damage. Attackers can broadly be characterized according to a number of attributes:

- *resources:* Resources include funds, personnel, and the skill levels of those personnel.
- *time:* An attacker may have very-near-term objectives or may be very patient and wait for an opportunity.
- *tools:* The sophisticated attacker can tailor attack tools to change his or her signature to avoid detection, or can develop tools or an email virus to target a specific system.
- *risk:* An attacker may seek publicity. An attacker operating outside of the United States may not be threatened by legal actions.
- *access:* Intruder access can be described in terms of the access mechanisms used in the attack, such as a dialup modem, a digital subscriber line (DSL), or the Internet; the origination of the attack, such as outside of a firewall, on a LAN, or connected from a trusted site; or the organizational position of the attacker, if any, such as employee, system administrator, or contractor.
- *objectives:* An attacker's objectives may include political, financial, criminal, military, and personal motivations.

The attributes of the attacker will affect the type of reasoning framework and the tactics used. For example, the preventative tactics appropriate for an unskilled user of a popular attacker toolkit may not be adequate for an expert-designed scenario. In addition, a system may have different modes of operation (e.g., peak vs. low demand, normal operation vs. maintenance mode). These different modes of operation may indicate the types of attacks likely.

A stimulated artifact: The target of the attack is of particular interest. For example, stronger encryption is required to protect a document than to protect the network traffic for a short-lived communication session, as an attacker has significantly more time to decrypt the file.

Response (the activity undertaken after the arrival of the stimulus): A security or survivability response will be to either prevent the attack from happening (passive) or to detect the attack and try to recover from its negative consequences (active). The choice of a response must be consistent with the risk mitigation strategies and constraints derived from the organizational risk and security analysis. The response objective could be prevention or, where prevention is not assured, containment of the impact and a rapid recovery to normal operations.

Response measure (the attribute-specific constraint that must be satisfied by the response): Security or survivability response measures will typically measure the reduced probability of attack or the reduced impact due to detection and recovery from

the attack. Usability or system administrative metrics might apply for user authentication preventative measures such as smart cards.

A set of architectural tactics: There are many sources in published literature of security techniques. While these works are too numerous to list in full, a few efforts are particularly noteworthy. For example, there is a community of software developers that is assembling an increasingly comprehensive collection of design patterns for information security [SecPat 04, Schumacher 03]. Others are collecting expert knowledge regarding general principles for information security [ISSA 04]. In addition, the RAND Corporation has published a method for improving the survivability of systems based on categories of predefined survivability vulnerabilities and techniques [Anderson 99]. Although RAND's method has not been applied extensively, the study surveyed a wide range of existing systems and research efforts on security and survivability to derive vulnerability and technique categories. Other work on survivability architectures also serves as a useful source of potential security and survivability strategies [Knight 00, Neumann 00].

The published techniques include

- *redundancy* – Anderson defines redundancy as "maintaining a depth of spare components or duplicated information to replace damaged or compromised assets" [Anderson 99]. Replicating components, connections, and/or data, often not co-located with the original copy, combined with good replication management can allow continued service when the original copy fails or is compromised.
- *diversity* – Diversity involves the use of different methods, components, and/or platforms to prevent attackers from exploiting the same vulnerabilities repeatedly. Examples include the use of different hardware, different operating systems, or even different programming techniques such as n-version programming. When such diversity is used at different system entry points it can increase the attacker work factor.
- *deception* – Deception can be used by the defender as well as a survivability threat that can be used by an adversary. Anderson defines deception, as it pertains to ensuring survivability, as an "artifice aimed at inducing enemy behaviors that may be exploited" [Anderson 99]. The most common example is the use of a collection of misinformation to waste an attacker's time as other mechanisms mount an appropriate response to the attack. This misinformation is often euphemistically referred to as a honeypot.
- *identification/authentication* – The NSA defines authentication as the verification of a claimed identity as legitimate and belonging to the claimant [NCSC 91]. Most types of access control require accurate identification and authentication of users. The most common technique used by far is a username and password. Stronger techniques using biometrics, tokens, and cryptographic signatures are also possible.
- *intrusion detection* – Intrusions require "both an overt act by an attacker and a manifestation, observable by the intended victim, that results from that act" [McHugh

00]. The goal of intrusion detection is to observe and report on the manifestation of an attacker's intrusion. Reporting can occur manually, through human analysis, or automatically using intrusion detection systems. Intrusion detection can take place in real time or through the offline analysis of system activity audit data recorded separately. Intrusions may be detected by looking for signatures of known attacks (virus checking is a common example) or by looking for anomalies—system activity that does not fit "normal" usage patterns. Available intrusion detection systems target either low-level network traffic or higher level application usage [McHugh 00]. Integrity-checking system executables based on expected parameters are also a form of intrusion detection.

- *recovery/adaptation* – Recovery and adaptation are the system responses to intrusion detection. Recovery is typically the near-term repair or replacement of data, components, or communications damaged due to intrusion. Adaptation typically involves longer-term planning and reconfiguration to prevent similar intrusions in the future. Examples range from complex techniques such as dynamic resource allocation to high-priority assets and activities and self-organization of distributed autonomous agents [Anderson 99] to simple techniques such as restoration from stored backups and error correction.
- *physical, logical, cryptographic, and temporal separation* – The security community has long regarded separation as fundamental to providing information security. Rushby and Randell first introduced these four primary types of security-related separation [Rushby 83], primarily as a means to separate entities of different classification levels. These strategies have also proven useful for information integrity and availability. The oldest strategy, physical separation, promotes security using spatial distribution and physical security mechanisms [NCSC 88], such as reinforced buildings, locks, and various types of shielding. Logical separation uses software-based mechanisms, such as message filters, functional wrappers, and security kernels, to control access. Cryptographic separation uses encryption and key management to protect data confidentiality and to detect data corruption to a degree proportional to the strength of the cryptographic algorithm and of the protection of private keys. Finally, temporal separation separates critical functions' execution in time. It is most closely associated with periods processing, "the processing of various levels of sensitive information at distinctly different times" [NCSC 88].
- *personnel management* – The security and survivability of any mission depends greatly on the trustworthiness, knowledge, and capability of the people in charge of mission support or execution. Trustworthiness is typically assessed through personnel security procedures [NCSC 88], such as periodic investigation of the backgrounds of people who have mission responsibilities. Ongoing performance appraisals are often a necessary complement to such investigations and provide additional information in terms of an individual's understanding and ability to perform the job adequately. Periodic training is also important to educate people on the role their jobs play in successfully achieving mission goals, the importance of security policy and procedures, and the possible impacts of inadequate performance.

2.2 Types of Security and Survivability Evaluators

Just as there are general and concrete security and survivability scenarios, there are general and specific security and survivability evaluators. The general evaluators help develop an integrated security and survivability strategy for a system that is consistent with its operating environment. The specific evaluators support reasoning about specific attack patterns and their mitigating security and survivability tactics. As one might expect, the research on general evaluators for security and survivability is much less mature than that for specific evaluators. This section discusses both, but the reasoning frameworks examples described in Section 4 only deal with specific evaluators.

2.2.1 General Security and Survivability Evaluators

Most of the work in the area of general security and survivability evaluators is in the area of security risk analysis, including the areas of adversary modeling, attack specification, vulnerability/threat analysis, security-related taxonomies and databases, impact analysis, and red teaming. Security risk analysis involves the analysis of system threats and vulnerabilities and their potential impact on the system's mission. The three primary elements of risk can be defined as follows [DoD 99b, DoD 00]:

- threat: any circumstance or event with the potential to cause harm to a system
- vulnerability: a system characteristic that could be exploited by a threat to harm a system
- impact: the extent of harm to a system that results from a threat's exploitation of a system vulnerability

Risk is formally defined as "a combination of the likelihood that a threat will occur, the likelihood that a threat occurrence will result in an adverse impact, and the severity of the resulting impact" [DoD 99a]. A malicious (intentional) threat can be viewed as any activity that purposely exploits a vulnerability in a system and results in a negative impact on mission success.

General evaluators, through such security risk analysis, identify the operational risks associated with the business, likely attacker motivations, and the impacts of general types of attacks on the business mission. The desired responses to those threats might reflect business metrics such as revenues, administrative costs, or the liability associated with contractual or regulatory requirements. The tradeoff between lower costs and more effective responses should be captured as the residual risk that the business stakeholders are willing to accept. In terms of quality-attribute-based architectural design, general evaluators would lead to a security and survivability strategy identifying a collection of concrete scenarios as requirements for the system to be built.

Experience with security risk analysis suggests a number of pitfalls to avoid in defining general evaluators for security and survivability [Soo Hoo 00]:

- *complexity* – Techniques often require explicitly considering all threats and vulnerabilities, from the most common to the most obscure, without some screening with regard to likelihood or impact. The resulting complexity tends to overwhelm the analysis.
- *incompleteness* – Techniques often ignore key aspects of the risk management problem or make incorrect assumptions about the problem domain. This may, for example, result in technological threats or solutions being emphasized over procedural ones.
- *data unavailability* – Techniques often require obtaining precise, quantitative data on the likelihood of threats and the severity of impact. In the real world, such data continues to be inconsistently collected and reported, and highly uncertain even when it is. Using highly uncertain "estimates" in places where precise data is required often leads to obviously faulty results or, even worse, to very misleading, but plausible, nonsense.
- *threat/countermeasure decoupling* – Techniques of managing security risk solely through the use of popular security technology and practices without a link to the mission objectives or threats tend to decouple the countermeasures from the risk they are supposed to reduce. This lack of traceability makes it difficult to accurately assess the actual residual risk resulting from the use of technology and practices.
- *static analysis* – Techniques generally deal only with the current threat environment, with little regard to managing the system under changing threats. Increasingly rapid changes in the threat environment, which are characteristic of modern Internet-based systems, demand techniques that can be applied as part of an evolutionary design and maintenance life cycle.

There are, of course, no easy solutions to these problems. Early research in security risk analysis generally promoted comprehensive solutions that became overly complex. More recent approaches simplified the methods at the expense of completeness [Soo Hoo 00].

2.2.2 Specific Security and Survivability Evaluators

As described above, the specific evaluators support reasoning about specific attack patterns and their mitigating security and survivability tactics, which results in scenario-specific reasoning frameworks. A scenario-specific reasoning framework supports the identification of security and survivability tactics for specific attack patterns. For example, a scenario that involves inappropriate access to information would lead to the analysis of the applicable access control tactics. The operational environment for a large geographically distributed organization might lead to an architecture that uses directory services to maintain the consistency and integrity of the authentication and access control data across multiple platforms and locations.

3 Types of Security and Survivability Reasoning Frameworks

A reasoning framework supports a design process that satisfies a single quality attribute. A reasoning framework is associated with a set of general scenarios, which are a precise system-independent specification of a collection of quality attribute requirements. Our discussion of security and survivability reasoning frameworks keys on the stimulus and response components of a general scenario. The stimulus or trigger is a condition that has to be considered when it occurs. A security stimulus can be a run-time event such as an attacker action, an operational event such as a change in a user's password, or a development event such as a change in a commercial component that impacts its integration into the system. The response component of a scenario describes the desired system behavior in response to the event. A general scenario also includes response measures.

Privacy or confidentiality requirements could be described by a scenario where the event is an attempt to access or change information, and the response describes the rules that govern access and any other actions associated with permitting or denying the access such as logging the event and the system's response. The response measures for an access or user authentication scenario include the confidence associated with correctly identifying the user, the time required to update user permissions on one or more systems, and usability metrics for both end users and system administrators. A reasoning framework also incorporates one or more quality attribute models. The model associated with privacy requirements might include the factors that impact system administration, such as the number of users, whether the information is accessed through multiple applications, and multiple storage locations for such information. The tactics associated with the reasoning framework provide the means to affect key parameters. For example, role-based access control is a tactic that can reduce the costs and complexity of managing access for a large and likely distributed user community. System management could be simplified and confidence in the enforcement increased by incorporating the privacy rules into a data service that is then used by the applications.

We begin our categorization by differentiating between *passive* and *active responses*. The objective for a passive response is prevention, containment, or protection. A passive response has the same behavior for all operating conditions. The use of operating system passwords and access control lists to meet confidentiality requirements is an example of a passive response. The validation of the input stream to an application is a passive response to a significant number of attacks. A firewall can provide a passive response to the probing of the network or applications to gather information to plan an attack. A passive response does not imply passive system administration. Effective user authentication may require rapid synchronization of user capabilities across all platforms, and some email virus protection

products require the capability to quickly update the necessary virus-signature descriptions on all clients.

An active response starts with the detection of attack and then describes an operationally-sensitive response to the trigger, as well as the eventual recovery of full services. A response could include changes to network or system configurations to manage a denial-of-service attack. Unusual activity observed for a specific peer in a distributed system might temporarily limit participation by that node until its status is clarified. Credit card fraud management is an example of an active response where unusual charges might require verification of those actions with the card owner before additional charges are permitted.

We begin by discussing passive frameworks.

3.1 Frameworks Based on a Passive Response

Security scenarios can also represent classes of attacks. The triggers may be steps in an attack or the consequences of an attack. The objective of a tactic might be to prevent known attack techniques, restore data integrity following a successful exploit, or to contain the impact of an active attack. While there is significant variation in the details of specific attacks, it is the common aspects of attacks that provide the most insight in directing survivable system development.

For example, many attacks share requirements to identify user accounts or to sketch the topology of the network that supports the workflow. Attacks can be categorized in terms of the kind of access and privileges required to execute the attack: user privileges are typically required for access to protected applications or data; system privileges are usually required to compromise logs to disrupt forensics; and network access is required to probe the network to identify available and vulnerable services. In addition, while vulnerabilities are often thought of in terms of the flaws of low-level components, vulnerabilities at an architectural level may be a much higher threat to an organization’s mission. In general, vulnerabilities may be apparent in human operations, the architecture of the technology, or individual technical components.

Consider a general scenario for a denial-of-service attack (DoS). Such an attack exhausts a system resource, but the tactic and the attribute model are influenced by the targeted resource. A network-based DoS could attempt to overload the network transport, exploit a network-protocol vulnerability, or target an essential component such as router or firewall. The reasoning framework should focus on the network architecture and tactics such as a firewall and network intrusion detection. An application-based DoS might use a vulnerability on a server to repeatedly crash the application or a required service. The tactics associated with the reasoning framework might include hardening the host or incorporating redundancy for essential services. Finally, a data-driven DoS could exceed the processing capacity of the data server or purposely insert sufficient data errors to exceed either the storage or processing

capacity of the logging component. Tactics for these triggers could increase capacity or use a run-time filter to quarantine the offending transactions.

The DoS examples also demonstrate that the tactics associated with a security reasoning framework are not just the classic security controls such as routers, firewalls, password authentication, and encryption, but also tactics associated with other quality attributes. Availability can require improved performance, while integrity is closely coupled with reliability. A critical observation is that the tactics selected to meet other requirements may also increase the opportunity for an attack. An architecture that uses a central data store might be a target for a host-based DoS attack, while an architecture that uses distributed and synchronized data stores might be impacted by a network-based DoS attack. An attack that targets a multi-organization business process could first compromise the system for one participant and then exploit any assumed trust among the cooperating systems to attack the other members.

Our preliminary categorization of reasoning frameworks has two axes. We will use the artifact targeted by the trigger and the characteristics of the response as the two axes. With respect to the artifact, we have

- *network-based attacks*: attacks on communication infrastructure and supporting services. Examples include network-based DoS attacks, including DDoS.
- *application-based attacks*: attacks on the architecture component applications, such as a Web server, email services, or supporting application infrastructure. Examples include exploits that target vulnerabilities of a Web server, such as a buffer overflow vulnerability, to gain increased access.
- *data-centered attacks*: attacks on the data stream or content presented by transactions. Such attack patterns can exploit or corrupt data and services or disrupt or deny essential services. Examples include attacks that target trust relationships between different machines or that target the gullibility of users (such as email attachments that contain malicious code).

The underlying models associated with a reasoning framework are also very dependent on the desired response. With respect to the response we have

- prevention. How are attacks prevented from gaining access to the artifact?
- containment. Assume an attack has not been prevented. How do we limit its scope?
- protection. What special facilities are used to protect especially vulnerable components?

Table 1 categorizes the reasoning frameworks based on these three criteria. The cells enumerate the tactics included in the reasoning frameworks. The selection of the tactics associated with a preventative response may need to model attacker behavior. A preventative

tactic may only need to increase the risk of detection or increase the demands on the attacker in terms of the skills or computing resources required.

The security tactics used in software developed by a third party may not reflect the actual threats associated with the intended operating environment. The security requirements for commercial software typically reflect general usage. Furthermore, externally-developed applications may have unknown vulnerabilities that cannot be eliminated. The requirements for legacy applications may not have considered Internet access. The prevention-based tactics represent the shared security assumptions among components. The protection-based tactics manage the exceptions when the security assumptions for a component are unknown or where the known behavior for a component may be an exploitable vulnerability.

	Network Attacks	Data-Centered Attacks	Application-Specific Attacks	User Input Attacks
Prevention	Insert a firewall that monitors traffic for network protocol vulnerabilities Insert a firewall that restricts access to vulnerable components such as a commercial off-the-shelf (COTS) application Insert a firewall that inhibits port scans and other attacker reengineering activities Encrypt communication	Insert an application proxy that filters input and output data streams Perform email virus scans Add hash functions (redundancy) to ensure integrity of data exchange between components Encrypt access control and authentication information to prevent user impersonation	Use a specialized operating system or required service to harden the server Limit available services to reduce exposure to attack	Implement signing, authentication, and access control to prevent impersonation of the user or providing a false source of information
Containment	Utilize a virtual local area network: This tactic partitions a large LAN into managed subnets. This limits the impact of an attack on a subnet or components of a subnet	Reduce privileges so that the system operates with the least privileges necessary. An intermediate attack objective is often to spawn a command shell that has the same access rights as the compromised component. Encrypt stored data Limit trust assumed when data exchanged among components	Utilize a demilitarized zone (DMZ) to isolate the exposed server Reduce the privileges necessary to operate the OS and infrastructure services to the least privileges necessary Maintain essential services on different hosts	
Protection		Add an application proxy that filters input and output data streams Insert a firewall that filters protocol and ports Insert a portal for authentication and access control		

Table 1: Categorization of Responses Based on Response Class and Artifact Under Attack

3.2 Frameworks Based on an Active Response

The use of an active response assumes that prevention, containment, and protection have failed. The objective of an active response is to tolerate or survive an attack by continuing operations. The trigger for the use of an active response is that an attack is recognized as being in process.

Active responses often consider an attack as a fault and think about failure recovery [Knight 04]. That is, what are the parallels between security and fault tolerance? The analysis required for an active security response and that for fault tolerance share some common objectives, including

- the need to analyze consequences of component failures, including failures from unknown causes. However, an attack can purposely inject multiple faults, such as disabling both an essential service and the recovery mechanisms. Reliability analysis often assumes independent faults.
- the focus on potential failures whose effects are likely severe or catastrophic. However, it is difficult to quantify the probability of a failure when those failures are caused by an intentional act. There is no metric equivalent to a mean-time-to-failure measure for hardware failures. While severe or catastrophic attacks can be speculated, the probabilities of such events may be insignificant. The probabilities may not depend on system attributes but rather on the likelihood of personnel, political, or financial circumstances that might motivate a skilled attacker to act.
- the common need to identify tactics that support the detection, containment, and recovery from such faults

However there are also some critical differences in detailed analysis.

- While detecting a security fault corresponds to sensing a software fault for reliability, a skilled attacker may be able to identify the security detection mechanisms employed and tailor the attack appropriately to avoid detection. This punch and counter-punch response pattern exists now between those who write email viruses and those who design the detection algorithms for email scanners. Detection may attempt to identify abnormal behavior, but normal system usage may evolve over time and require continual changes to the criteria. Unusual activity is often generated by faults not associated with an attack.
- The recovery services could be a target of an attack. Hence the recovery procedures must also be survivable. Security may be able to take advantage of redundancy introduced for reliability or for managing significant variations in normal system loading. An active response to a DoS attack could redeploy any reserve capacity to maintain the required level of service.
- Security and reliability reasoning frameworks for the control structures address similar design issues. The control decisions may involve balancing autonomy for applications or hosts versus system-wide coordination of the response or determining the visibility of the

class of faults. The control architecture may be driven by or drive the overall control model: peer-to-peer versus hierarchical.

The model associated with an active response could be thought of as a control system [Sullivan 99], and a reasoning framework could be structured to support the design of such a control system. The sensors in such a control system could include an intrusion detection system, as well as system monitors to identify abnormal activity.

The characteristics of the response are significant design drivers. The parameters that influence the choice of tactics depend on the impact and target of the attack—the network bandwidth, data storage, or the processing capacity. Can the impact of an attack be contained by limiting the scope of an attack? Is the impact associated with a specific workflow, with a class of users, with a physical location, or with a specific platform? Workflow controls might require the capability to dynamically reconfigure applications, and the containment would be complex if there are dependencies among workflows. An active response might be to reduce functionality by providing only the essential services. The operational security policy, with respect to the access allowed or the authentication required, could be dynamically changed. While the capability to contain an attack may permit continuing operations for those activities that are not at risk, the diversity of controls increases system complexity.

The limited experience with and complexities associated with an active response suggest that it should be approached with caution and the scope of its application controlled. An active response suggests the need for an adaptive architecture, but the design of such architecture is still a research topic. An active response requires an effective intrusion detection component, as activity improperly identified as an attack might force an unnecessary system reconfiguration. The worm propagation example in Section 4 uses an active response for the containment of an attack. The containment tactic is to slow rather than stop selected communications, which reduces the effects of a false positive by the attack sensors.

The analysis for an active response should draw on the reliability analysis. Single points of failure must be both reliable and secure. Extensive shared state among components complicates recovery from a hardware or software fault and limits the ability to contain an attack. The shared security and reliability analysis might lead to design guidance to limit the state that is maintained by a component or the state shared among components in order to limit the impact of an attack and reduce the scope of the required recovery.

4 Examples

The three examples in this section are representative of existing security reasoning frameworks.

The first example considers active responses to a DDoS attack. While the analysis for an active response is usually more complex than that for a passive response, the controlled scope of the response enables well-defined reasoning frameworks. The examples illustrate countermeasures for both the ultimate traffic flooding of the victim and the limitation of any self-propagating worms that might lay the DDoS agent infrastructure.

The second example analyzes a design for a specific type of vulnerability called a covert channel. The primary objective of the analysis is to demonstrate that it would be difficult to exploit a design to obtain undesired access to sensitive information and not to identify preventative tactics; hence, it is not a complete reasoning framework. It may be possible to refine the vulnerability analysis to provide better design guidance.

The access management tactics are passive and hence should be representative of a mature reasoning framework. But access management, the last example, may be the least satisfactory of the three examples. Many of the key parameters for the attribute model are organizational rather than technical. The scope is the design of an integrated access control management system that supports multiple applications, locations, work processes, and possibly organizations. Those requirements and the emerging technologies designed to support them stretch if not break the reasoning frameworks that are applicable for a single application or platform.

4.1 Reasoning Frameworks for Inhibiting DDoS Attacks

We are concerned in this section with the following trigger conditions:

- stimulus and source of stimulus: An attacker attempts to disable a victim computer through flooding the victim with Internet messages.
- response and response measure: The traffic through computers attached to the Internet is restricted so that abnormal traffic is metered but normal traffic is unaffected.

DDoS attacks depend on the attacker’s control of many computers spread around the Internet IP address space. The attacker recruits these controlled computers, which we call DDoS agents, through automatic scanning of remote machines looking for vulnerabilities that enable subversion. In addition, DDoS agents are themselves used to recruit other DDoS

agents, often using self-replicating computer worms to create the DDoS agent network. The attacker coordinates the agents to individually but simultaneously perform DoS attacks on some organization's Web server after assembling a sufficient arsenal. As depicted in Figure 1, attack coordination is performed using control traffic via a set of compromised hosts called handlers. The flood traffic targeting the victim organization's Web server prevents the delivery of service if the number of agents and the size of the DoS attack delivered by each agent are sufficiently large.

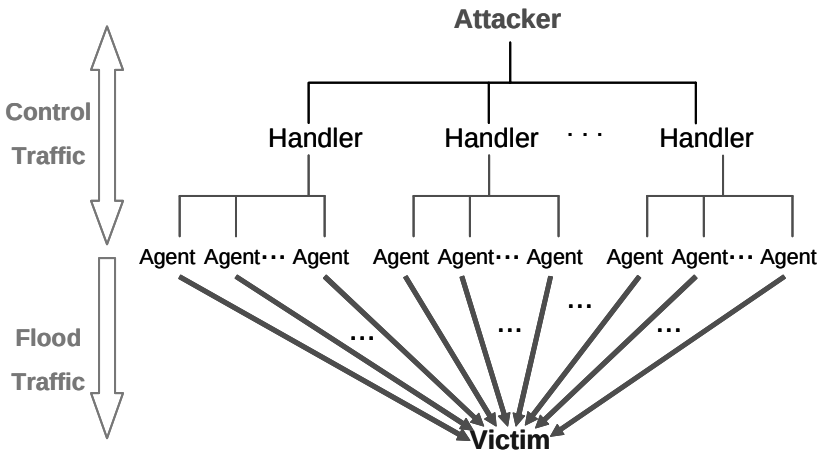

Figure 1: DDoS Attack Infrastructure

Preventing or inhibiting DDoS attacks requires a multidimensional approach. Since DDoS attacks require a very large number of DDoS agents to be effective, DDoS agent infrastructures are typically set up using self-replicating computer worms. Thus, tactics for inhibiting worm propagation can in effect inhibit DDoS attacks by inhibiting the growth of the DDoS agent infrastructure. This requires the active participation by parties that are not directly or not significantly affected by the DDoS attack, i.e., organizations that have vulnerable or compromised computers that may serve as a DDoS agent. This is clearly not a complete defense. Tactics are also needed that assume that the DDoS agent infrastructure is already in place but that inhibit the flooding of the victim organization's server by the DDoS agents. The rest of this section provides examples of these two types of tactics.

Both examples pertain to any Internet service provider that provides service to organizations that are high profile (e.g., the U.S. government), susceptible to embarrassment from computer attacks (e.g., Microsoft), or whose mission depends heavily on the availability of their Web services to external customers (e.g., Amazon or eBay). Organizations that support high-volume Web traffic need a powerful Web service capability. Such organizations are attractive targets for worm propagation and to be used as a DDoS agent that targets other organizations. Being used in this way, an organization could suffer embarrassment, loss of customer confidence, and legal liability.

4.1.1 A Reasoning Framework for Inhibiting Worm Propagation

Worm propagation can be inhibited by constraining the IP traffic exiting the organization's local operating domain to limit gross violations of locality [McHugh 03, Williamson 02]. The assumption is that legitimate outgoing IP traffic for the organization is going to be confined to a limited range of IP address destinations—the program's working set—considered to be *local* for that organization. The primary tactic then is to use a soft limit to react to locality violations:

> *When a program requests an IP address that is not in that program's working set, "the access request is placed in a paced delay queue which limits the rate at which such requests can be dispatched to one request per unit delay. When a delayed access is dispatched, all queued pending requests destined for the same address are sent immediately in the order in which they were enqueued and the destination is inserted in the working set, replacing its least recently accessed address. In this way small locality violations are tolerated with minimal delay, but gross violations encounter ever increasing delays."* [McHugh 03, Williamson 02]

There are two possible options for the measuring locality. In the first option, each program generating outgoing IP traffic has a fixed size working set, so that when a new address is seen, the least recently used address is replaced with the newly referenced one. In this case, locality is measured as the frequency with which the working set changes. In the second option, each program generating outgoing IP traffic has a variable-size working set with constant removal, so that new addresses are added and least recently used entries are removed at fixed intervals. In this case, locality is measured as the current size of the working set.

The evaluator for the implicit reasoning framework within which this tactic resides requires that the developer identify

- IP address profiles for each Web browsing program within the local operating domain
- the best representation of the working set (i.e., option 1 or option 2 above) and decide how the working set is allowed to change over time

Through experimentation, the developer must decide on the frequency with which nonlocal addresses may be accessed that slows the propagation of worms sufficiently and leads to an acceptable false positive rate. Independent parameters for the quality attribute model include

- each Web browsing program's working set
- the representation of the working set
- how the working set is allowed to change over time

Dependent parameters include

- the frequency of nonlocal IP address accesses permitted
- the false positive rate (i.e., limiting the unnecessary slow-down of legitimate Web traffic)
- the slow-down of malicious traffic required

4.1.2 A Reasoning Framework for inhibiting DDoS Flood Traffic

DDoS flood traffic can be inhibited by filtering or rate limiting the IP traffic that deviates from the expected entropy of normal IP traffic [Feinstein 03]. Entropy is computed as a measure of the randomness across some sample of packet header fields. Comparing the entropy of one sample of packet headers with another sample enables the detection of changes in the randomness of IP traffic. Experimentation has shown that the entropy of normal traffic falls within a narrow range of values, whereas the entropy of DDoS attack traffic exceeds these ranges in a detectable manner. The primary tactic then is to filter or rate limit traffic that exceeds the normal traffic entropy range.

The evaluator for the implicit reasoning framework within which this tactic resides requires that the developer identify

- the set of symbols in each packet to be used to compute entropy
- the number of sequential packets to be used in the computation, i.e., the window size

Through experimentation, the developer must decide on the window size that sufficiently slows the DDoS attack traffic and leads to an acceptable false positive rate. Feinstein et al. describe the window size as a “tunable parameter that controls how much smoothing of short-term fluctuations the detector will do” [Feinstein 03]. Large window sizes keep the expected variation in entropy small and lead to low false positive rates due to brief and, presumably, insignificant anomalies. However, there is a tradeoff, since the larger the window size, the slower the detection of real attacks. Feinstein et al. have found through experimentation that a window size of 10,000 packets is a reasonable compromise.

Independent parameters for the quality attribute model include

- the window size of the packet stream used to calculate entropy
- the entropy value range for normal IP traffic
- the entropy values of incoming IP traffic

Dependent parameters include

- the deviation from normal that justifies action (i.e., to limit attack traffic)
- the acceptable false positive rate (i.e., limiting the unnecessary slow-down or filtering of legitimate Web traffic)

- the slow-down of attack traffic required

Of course there are limitations to this tactic. Feinstein et al. describe an important limitation as follows:

> *"A sophisticated attacker would likely attempt to defeat the detection algorithm by creating stealthy traffic floods that mimic the legitimate traffic the detector would expect. An attacker who knew that the entropy of various packet attributes was being monitored could build an attack tool that generates floods with tunable entropy levels. Through guesswork, penetration, or trial and error, the attacker could determine typical entropy levels seen at the detector and tune the flood to match"* [Feinstein 03].

The authors propose that defeating the purpose of this tactic in this way may be prevented by introducing multiple detectors between the flood source and target, since expected entropy ranges are likely to differ in different network environments.

4.2 Prevention-Based Tactics for Inhibiting Information Leakage

We are concerned in this section with the following trigger conditions:

- an organization has sensitive (possibly classified) information
- no information (or limited information) should be accessible to unauthorized individuals or other systems

Information leakage is the term used to describe such trigger conditions. The leakage may be due to malicious software such as a Trojan horse program. Organizations that own highly classified information resident on systems that must connect (even if only indirectly) to other untrusted systems or networks, such as the Department of Defense, are often concerned with understanding the potential for covert information leakage. This concern is seen in the covert channel analysis requirements of security evaluation standards for higher assurance levels, such as the Common Criteria.

Information leakage (or information sharing, in general) between parties is accomplished through resources shared by those parties. There must be at least one sending process and at least one receiving process. The sender must be the holder of the sensitive information that the organization wants to keep confidential, i.e., to prevent the receiver from obtaining.

A concrete example of a system component for which information leakage might pose a serious problem is a database replication component that transfers critical data from an inventory database at a logistics coordinator to a database at the commander-in-chief (CINC) headquarters for military planning purposes. A common requirement in such a situation might be that the maximum capacity of any signaling channel through the data replication

component from the CINC headquarters database to the inventory database must be fewer than 100 bits per second. In this example, the CINC headquarters database presumably contains much more classified information than is permitted on the logistics coordinator's systems.

Before deciding on which tactics are appropriate to inhibit covert channels, one needs to identify what covert channels exist and the information-carrying capacity of those covert channels. The Shared Resource Matrix (SRM) method was developed for exactly this purpose [Kemmerer 02]. The SRM method assumes a single processor system that is shared by multiple processes with the following constraints:

- Processes have access to a set of operations.
- Each operation may access a set of resources.
- Each resource has a set of attributes.
- A shared resource is any object or collection of objects that may be referenced or modified by more than one process.
- Each process may **reference** or **modify** only a subset of the attributes for a given resource.
- Processes may be local or distributed, but only one process may be active at any one time.
- Processes can send or signal information to other processes if permitted by a defined security policy.

In abstract, the SRM method helps to identify attributes of shared resources that a sending process can modify and a receiving process can reference (directly or indirectly) so as to signal information in violation of a defined security policy. The SRM method also helps to quantify the rate of that signaling.

In a bit more detail, an SRM comprises

- the set of primitive operations of the system being analyzed listed as the matrix columns
- the set of attributes of all resources that may be shared by the sender and receiver listed as the matrix rows
- indications as to whether each primitive references or modifies each attribute indicated as an "R" or an "M" in each matrix cell

These elements, plus the security policy defining permitted interprocess communication, are the independent parameters of the quality attribute model provided by the SRM method.

Analysis of the SRM proceeds by constructing the transitive closure of the SRM, since the read of an attribute in one operation, when that attribute has been modified in a second

operation, is an indirect read of all the attributes that contributed to the modification. The analyst must then determine whether

- the sending and receiving processes can have access to the same attribute
- and there is some means by which the sending process can force the shared attribute to change
- and there is some means by which the receiving process can detect the attribute change

If these conditions are satisfied, the analyst needs to find some mechanism for initiating the communication between the sending and receiving process and for sequencing the events correctly. This mechanism could be another channel with a smaller bandwidth. If the analyst can find such a mechanism, he or she must determine which of the four cases apply:

- Another legal or overt channel operates between the sender and receiver, so the channel is of no consequence.
- No useful information can be gained from the channel. This will be the case if the only information that can be signaled over the channel is information that the receiver already possesses.
- The sending and receiving processes are the same.
- A covert channel exists and should be analyzed further.

For those information flows in the last case, the analyst must determine the capacity of the covert channel by quantifying

- the quantity of information that can be transmitted per execution of a scenario. Usually this is quantified in number of bits as determined by the SRM analysis.
- the time required to exercise the scenario. This may be determined experimentally, but is highly dependent on the speed of the processor.
- the effect of other system activities on the effectiveness of the transfer. The conservative approach is to assume no degradation.

These elements are the dependent parameters of the quality attribute model defined by the SRM method. The method itself is the “evaluator” of the reasoning framework.

Tactics for inhibiting channels that are in excess of that allowed by the security policy include

- adding noise to the channel, e.g., by adding artificial activity to the process
- restricting the amount of information carried per scenario
- restructuring the system to eliminate or inhibit the covert channel, e.g., by using sandboxing or functional wrapping technologies

- increasing the time required to execute the scenario. This is particularly useful when the scenario is intended for execution by people, in which case execution times can be increased to prevent malicious spoofing at electronic speeds.

Detect and respond approaches may also be useful to audit the use of covert channel leakage and respond to them on a case-by-case basis. In extreme or suspicious cases, incident handling may involve disconnecting systems until a complete investigation can be conducted.

4.3 A Reasoning Framework for Access Management

We are concerned in this section with the following trigger conditions:

- a user who has no access privileges is to be assigned access privileges
- a user with existing access privileges is to have these privileges modified
- a user with existing access privileges is to have these privileges revoked

Each of these scenarios has a response measure that is in minutes rather than hours or days.

Managing access control permissions is a major problem for large, distributed computing environments. The examples in this section demonstrate some of the parameters that a reasoning framework for access management must include. Such a complete reasoning framework does not yet exist. We will briefly describe why access management is difficult, one of the tactics used for access management, why that tactic is not successful, and an alternative tactic that is being proposed.

The increasing number and complexity of identify and access scenarios are a consequence of multiple trends:

- increased integration of systems within organizations
- adoption of business practices that require customers, partners, suppliers, and employees to have access to information assets. The requirements may be derived from contractual agreements among the participants.
- enactment of privacy regulations for controlling access to medical, personal, or financial information: Health Information Portability and Accountability Act (HIPAA), Gramm-Leach-Bliley Act (GLB), and European Union Privacy

The changes in requirements are characterized by a growing number and type of users, applications, and access methods. The computing environments for the classic access control and authentication model were timeshares. While that model has evolved, the design assumed local autonomy for policy and administration, and each application, system, or platform developed its own proprietary mechanisms.

The most significant changes arise from the emerging requirements for maintaining the semantics associated with a security policy for a distributed system. The choice of the proper access control tactic depends on knowing the semantics associated with the data. A transfer of a portion of a patient's medical record also requires designating the access policy associated with such information and the enforcement of that policy by the receptor.

Many of the dependent parameters for access control will be drawn from business requirements. The scope and scale of the domain are critical parameters to a quality attribute model. The system administration procedures for tactics that assign permissions individually don't scale to organizations with users numbered in the ten thousands with an equally large number of assets distributed across hundreds of applications. The user community for a Web-based application might be in the millions.

Data management of the security metadata is another key parameter. Changes in permissions may require synchronization of user security information across multiple systems, while personnel actions such as terminations may require the capability to immediately disable access. There are commercial products that support what has been called provisioning, i.e., the distribution of the required security data to multiple hosts and applications.

The interoperability and data management requirements may be the most difficult to satisfy. Security policies are designed around the business work flows, and the security architecture may have to support an ever-changing set of work processes. The desired response time for implementing a new work process might be weeks or months. Work flows or security policies could be specified in a manner to enable a work flow or security policy to be defined at start-up or to enable a dynamic binding of a security policy for a specific data exchange.

The concept of roles and role-based access control (RBAC) is a well-known technique that is often used to reduce the system administrative load. With RBAC, a combination of organizational and system analysis identifies the operations that must be executed by persons in particular jobs and the computing privileges required for those activities. Security administration for RBAC consists of assigning employees to the proper roles. Specific applications or platforms may still require individual access control data, but provisioning software could map the privileges required by a role to application-specific access controls and manage the changes in roles or role assignments across multiple platforms and applications.

A successful application of RBAC often depends on managing the scope of the effort. There are usually conflicts among the organizational-defined roles, application constraints (mapping roles to application-specific access control lists), and operational constraints in how the work is really done. The actions authorized for a role may vary across the organizational units, be tailored for a specific work flow, or depend on the physical location of the individual. That diversity can lead to administrative difficulties equivalent to those in a design that manages access control individually. From one perspective, one RBAC objective

is to normalize roles, but such an objective may not be achievable for a large organization with frequently changing business practices. This is why scope must be a parameter for any quality attribute model that is a portion of an access management reasoning framework.

Another parameter should be the binding time for roles. For a diverse work environment, the dynamic binding of roles may reduce the need to define a significant number of static roles. Rather, a design could define a small number of generic roles that are statically assigned and then complete the binding at run time by incorporating operational information such as the location or the execution of a specific work activity. Some roles may not be static. Consider the application of RBAC to hospital operations. An individual may be statically assigned the role of a nurse, but the enforcement of privacy regulations for access to medical records may require that the individual has access only if they have responsibility for some aspect of the patient's treatment. Access might require that the individual be currently on duty in a specific hospital unit.

Dynamic binding of roles requires secure and reliable management of the attributes that are used in the assignment. The dynamics of the access decision could become a vulnerability if those attributes could be compromised. The dynamic binding for roles may also generate auditing requirements. Static binding of roles might require auditing only for tools that administer those attributes. Dynamic binding of roles may require logging every assignment in order to monitor for unusual activity or to satisfy regulatory or contractual requirements.

A side effect of access management tactics may impact the required computing infrastructure and the operational implications. Design options, such as using directory services or provisioning software to maintain security metadata, may be costly and also may introduce additional vulnerabilities. A complex infrastructure may be an impediment to future system changes or be a continuing financial overhead because of the maintenance costs.

5 Conclusion

In this note, we have provided an introduction to a way of thinking about security. Our ultimate goal for the overall effort is that an analyst has a complete set of reasoning frameworks for what is commonly called security, and each of the reasoning frameworks has all of the portions we have identified. If this goal is achieved, then the software architect or security analyst has the material to ensure that the system being designed achieves specified goals with respect to security.

Many open issues remain between what we have provided here and our goal. These include the following:

- What is the complete set of reasoning frameworks relevant to security and how are they related? The significant variation in the details of specific attacks encourages a multiplicity of frameworks, but a specific attack technique often has a very short effective life as systems are patched to prevent the exploit. Reasoning frameworks should be based on the common objectives for a class of attacks. We provided an initial guess at a taxonomy for security reasoning frameworks, but how close our guess is to our goal is an open issue.
- The reasoning frameworks that are identified in this note are only partially complete. Each one of them must be made complete. The greatest difficulty in completing a reasoning framework lies in the definition of the evaluator. The evaluator defines the independent and dependent parameters that are crucial to disciplined thinking about security.
- A reasoning framework has to model both the quality attribute and the attacker. The list of security scenarios is not static. Security design is more like a game in which the defender and the attacker continually adjust to each other's tactics. Unfortunately, the defender has to plug all the holes, while the attacker has to only find one. A defense does not necessarily have to be perfect. It may be sufficient to just increase the cost to the attacker in terms of the skills, resources, or time required for an attack. But attack patterns and attacker characteristics are ever changing, and a design based on current attacker profiles may not be effective for the next generation of attack. How to manage the ever-changing attack patterns and attacker profiles is an open question. At a minimum, it may be wise to apply a classic modifiability tactic and isolate knowledge of attacker behavior to only a few components.
- The tactics must be described in more detail. In this note, we describe a tactic with a word or phrase. The meaning of this word or phrase is subject to interpretation.

- Regardless of these shortcomings, this note provides the beginnings of an approach to designing for security. Furthermore, this approach is being used for other quality attributes, such as performance and modifiability, that should enable consideration of tradeoffs in a systematic fashion as well.

References

URLs are valid as of the publication date of this document.

[Anderson 99] Anderson, R. H.; Feldman, P. M.; Gerwehr, S.; Houghton, B. K.; Mesic, R.; Pinder, J.; Rothenberg, J.; & Chiesa, J. R. "Securing the U.S. Defense Information Infrastructure: A Proposed Approach" (RAND Report MR-993-OSD/NSA/DARPA). Santa Monica, CA: RAND Corporation, 1999. http://www.rand.org/publications/MR/MR993 (1999).

[Bachmann 02] Bachmann, Felix; Bass, Len; & Klein, Mark. *Illuminating the Fundamental Contributors to Software Architecture Quality* (CMU/SEI-2002-TR-025, ADA407778). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002. http://www.sei.cmu.edu/publications/documents/02.reports/02tr025.html.

[Bachmann 03a] Bachmann, Felix; Bass, Len; & Klein, Mark. *Deriving Architectural Tactics: A Step Toward Methodical Architectural Design* (CMU/SEI-2003-TR-004, ADA413644). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003. http://www.sei.cmu.edu/publications/documents/03.reports/03tr004.html.

[Bachmann 03b] Bachmann, Felix; Bass, Len; & Klein, Mark. *Preliminary Design of ArchE: A Software Architecture Design Assistant* (CMU/SEI-2003-TR-021, ADA421618). Pittsburgh, PA; Software Engineering Institute, Carnegie Mellon University, 2003. http://www.sei.cmu.edu/publications/documents/03.reports/03tr021.html.

[Bass 01] Bass, L.; Klein, M.; & Moreno, G. *Applicability of General Scenarios to the Architecture Tradeoff Analysis Method* (CMU/SEI-2001-TR-014, ADA396098). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2001. http://www.sei.cmu.edu/publications/documents/01.reports/01tr014.html.

[Bass 03] Bass, L.; Clements, P.; & Kazman, R. *Software Architecture in Practice*, 2nd ed. Reading, MA: Addison-Wesley, 2003.

[DoD 99a] U.S. Department of Defense. *DoD Information Technology Security Certification and Accreditation Process (DITSCAP).* DoD Instruction 5200.40, November 30, 1999.

[DoD 99b] U.S. Department of Defense. "Introduction to Threats to Department of Defense Information Systems." *Secret and Below Interoperability Initiative Report*, September 30, 1999.

[DoD 00] U.S. Department of Defense. "Basic Risk Management for Department of Defense Information Systems: Informal Reference Guide Edition 1.1." *Secret and Below Interoperability Initiative Report*, January 21, 2000.

[Feinstein 03] Feinstein, L.; Schnackenberg, D.; Balupari, R.; & Kindred, D. "Statistical Approaches to DDoS Attack Detection and Response," 303-314. *Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX '03)*. Washington, D.C., April 22-24, 2003. Los Alamitos, CA: IEEE Computer Society, 2003.

[ISSA 04] Information Systems Security Association. "Generally Accepted Information Security Principles: GAISP V3.0." 2004. http://www.issa.org/gaisp/gaisp.html.

[Kemmerer 02] Kemmerer, Richard A. "A Practical Approach to Identifying Storage and Timing Channels: Twenty Years Later," 109-118. *Proceedings of the 18th Annual Computer Security Applications Conference (ACSAC)*. San Diego, CA, Dec. 9-13, 2002. http://www.acsac.org/.

[Knight 00] Knight, J. C.; Sullivan, K. J.; Elder, M. C.; & Wang, C. "Survivability Architectures: Issues and Approaches," 157-171. *Proceedings of the 2000 DARPA Information Survivability Conference and Exposition (DISCEX 2000), Vol. 2*. Hilton Head, SC, Jan. 25-27, 2000. Los Alamitos, CA: IEEE Computer Society, 2000.

[Knight 04] Knight, John C. & Strunk, Elisabeth A. *Achieving Critical System Survivability through Software Architectures. Architecting Dependable Systems*. Edited by R. de Lemos, C. Gacek, & A. Romanovsky. Heidelberg, Germany: Springer-Verlag, 2004.

[McHugh 00] McHugh, J.; Christie, A.; & Allen, J. “Defending Yourself: The Role of Intrusion Detection Systems.” *IEEE Software 17*, 5 (September/October 2000): 42-51.

[McHugh 03] McHugh, John & Gates, Carrie. “Locality: A New Paradigm for Thinking About Normal Behavior and Outsider Threat,” 3-10. *Proceedings of the 2003 Workshop on New Security Paradigms*. Ascona, Switzerland, August 18-21, 2003. New York: ACM Press, 2003.

[NCSC 88] National Computer Security Center. “Glossary of Computer Security Terms.” NCSC-TG-004 Version 1, October 1988.

[NCSC 91] National Computer Security Center. “A Guide to Understanding Identification and Authentication in Trusted Systems.” NCSC-TG-017 Version 1, September 1991.

[Neumann 00] Neumann, P. G. “Practical Architectures for Survivable Systems and Networks.” Menlo Park, CA: Computer Science Laboratory, SRI International, June 2000. http://www.csl.sri.com/neumann/survivability.pdf (2000).

[Rushby 83] Rushby, J. M. & Randell, B. “A Distributed Secure System,” 127-135. *Proceedings of the IEEE Symposium on Security and Privacy*. Oakland, CA, April 25-27, 1983. Maryland: IEEE Computer Society Press, 1984.

[SecPat 04] Security Patterns Homepage. http://www.securitypatterns.org/.

[Schumacher 03] Schumacher, M. *Security Engineering with Patterns: Origins, Theoretical Model, and New Applications*. Heidelberg, Germany: Springer-Verlag, 2003.

[Soo Hoo 00] Soo Hoo, K. J. “How Much Is Enough? A Risk-Management Approach to Computer Security.” *Report of the Consortium for Research on Information Security and Policy*. Center for International Security and Cooperation, Stanford University, June 2000.

[Sullivan 99] Sullivan, K., Knight, J. C.; Du, X.; & Geist, S. “Information Survivability Control Systems,” 184-192. *Proceedings of the 1999 International Conference on Software Engineering*. Los Angeles, CA, May 16-22, 1999. Los Alamitos, CA: IEEE Computer Society Press, 1999.

[Wallnau 03] Wallnau, K. *Volume III: A Technology for Predictable Assembly from Certifiable Components (PACC)* (CMU/SEI-2003-TR-009, ADA413574). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003. http://www.sei.cmu.edu /publications/documents/03.reports/03tr009.html.

[Williamson 02] Williamson, Matthew M. “Throttling Viruses: Restricting Propagation to Defeat Malicious Mobile Code.” *Proceedings of the Eighteenth Annual Computer Security Applications Conference*. San Diego, CA, December 9-13, 2002. http://www.acsac.org/.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	September 2004	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
Security and Survivability Reasoning Frameworks and Architectural Design Tactics	F19628-00-C-0003

6. AUTHOR(S)

Robert J. Ellison, Andrew P. Moore, Len Bass, Mark Klein, Felix Bachmann

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2004-TN-022

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116	

11. SUPPLEMENTARY NOTES

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

13. ABSTRACT (MAXIMUM 200 WORDS)

The Software Engineering Institute (SEI) has been investigating disciplined software architecture design for several years. The SEI approach includes a collection of "quality attribute reasoning frameworks" that understand both quality attribute reasoning and how architects design for the quality attribute under particular situations. The approach was first applied to the quality attributes of modifiability and performance. This report is an initial attempt to use the same method for the related quality attributes of security and survivability. The report includes an initial organization of security within the framework, a partial explication of elements of that framework, and three representative examples of existing security reasoning frameworks.

14. SUBJECT TERMS	15. NUMBER OF PAGES
security architecture, architecture design, security vulnerability, architecture tactic, security design	41

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