QUANTITATIVE FRAMEWORK TO DESIGN SERVICES WITH INTRUSION TOLERANT QOS

by

Quyen L. Nguyen
A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
in Partial Fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Computer Science

Committee:

_____ Dr. Arun Sood, Dissertation Director

_____ Dr. Duminda Wijesekera, Committee Member

_____ Dr. Sam Malek, Committee Member

_____ Dr. John Shortle, Committee Member

_____ Dr. Sanjeev Setia, Department Chair

_____ Dr. Kenneth S. Ball, Dean, Volgenau School of Engineering

Date:_____ Spring Semester 2014
George Mason University
Fairfax, VA

Quantitative Framework To Design Services with Intrusion Tolerant QoS

A Dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Quyen L. Nguyen
Master of Science in Computer Science
University of California at Berkeley, 1994
Bachelor of Science in Computer Information Science
University of Delaware, 1993

Director: Arun Sood, Professor
Computer Science Department

Spring Semester 2014
George Mason University
Fairfax, VA

DEDICATION

This dissertation is dedicated to my parents who have given me the best education during my childhood.

ACKNOWLEDGEMENTS

I am deeply grateful to Dr. Arun Sood for initiating me to the Intrusion Tolerance architecture. Dr. Sood has guided me all the way during my research, challenged me to come up with new ideas and concepts, and encouraged me to publish findings in magazines and conferences.

I would like to thank Dr. Duminda Wijesekera, Dr. Sam Malek, Dr. John Shortle, and Dr. Paul Sousa for their time serving on my dissertation committee. I also want to thank them for their extremely valuable comments, feedback, and advice that helped to improve this dissertation.

I would like to thank Dr. Domenico Ferrari for being my research advisor for my MS Research Project within the Tenet Group at UC Berkeley. I would like to express my appreciation to Dr. James Demmel for encouraging me to pursue academic research.

I would like to give my special thanks to Dr. David Wood for giving me the very first opportunity to be a research assistant after my freshman year, and Dr. David Saunders for being my research advisor for my BS Research Project at the University of Delaware.

I would like thank my peers, Ajay Nagarajan and Robert Banks, who have shared the common interest in Intrusion Tolerance as a mechanism to enhance security and resilience of the system architecture.

Most of all, I would like to thank my wife for supporting me during the years of my PhD study and research in Computer Science at George Mason University.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

QUANTITATIVE FRAMEWORK TO DESIGN SERVICES WITH INTRUSION TOLERANT QOS

Quyen L. Nguyen, Doctor of Philosophy

George Mason University, 2014

Thesis Director: Dr. Arun Sood

Large software systems can be designed as a set of loosely coupled services interacting with each other; simple services can be composed to form more complex services. But, for services to be usable in production, they must satisfy non-functional requirements, especially security-related quality of service in order to ensure confidentiality, integrity, and availability. Unfortunately, software vulnerabilities expose these services to malicious actors, and make them susceptible to attacks. Due to the distributed and decentralized nature of services, publishing and guaranteeing security quality of service are crucial so that potential applications and clients can make use of the provided services. On the other hand, intrusion prevention and detection are not perfect in securing services, due to the increased sophistication of malicious attacks. This has motivated the addition of the Intrusion Tolerant component to complement the line of defense for applications and services. Given the need of making services intrusion-tolerant, my

research focuses on providing an Quantitative Framework for Intrusion Tolerant Services (QFITS) for a systematic and quantitative approach to model, design and implement services with Intrusion Tolerant Quality of Service (IT-QoS). The approach relies on: a) the foundation of the architecture of Self Cleansing Intrusion Tolerance; b) a correlation component for which I will use Semi-Markov Model to compute IT-QoS metrics and then prove that there exists a mathematical dependency between those metrics and intrusion tolerance control parameters such as the exposure window in the case of a recovery-based architecture; c) a software specification mechanism which is based on a proposed Unified Modeling Language profile that allows software architects to model IT-QoS for services. To system architects of service providers, the framework would also constitute as the basis for ensuring differentiated levels of certain IT-QoS metrics such as Secure Availability, and Mean Time To Security Failure (MTTSF), which are indicators the reliability of a service operating in the presence of cybersecurity attacks.

CHAPTER 1 - INTRODUCTION

This Introduction chapter describes the motivation of my research and the contributions of the dissertation in the area of ensuring intrusion tolerance QoS for services in a Service-Oriented Architecture (SOA).

1.1 Motivation

Many enterprises have adopted the SOA paradigm, since it facilitates the design of a large system as a set of loosely coupled services. Furthermore, these services can be combined to form more complex services and applications. Web Service, whether SOAP-based or RESTful, is the prominent implementation for a service that utilizes open standards and Internet protocols. The benefits of Web Services within an SOA system have to be balanced with the security challenges. As with any software system, a service must satisfy non-functional requirements, and security quality in particular in order to ensure availability, integrity, and confidentiality. Guaranteeing security quality is even more challenging when services are deployed in distributed hosts within an enterprise network or over a wide-area network. Moreover, a recent trend in Information Technology is the utilization of services provided via Cloud Computing by entities external to an organization. Since a centralized quality control cannot be done, one needs QoS guarantees for the services participating in an application. Unfortunately, with the

proliferation of web services, security attacks have become more sophisticated in such a way that they can bypass traditional security protection measures such as firewalls and intrusion detection. Therefore, a system cannot rely solely on intrusion prevention and detection for its security protection, but should include Intrusion Tolerant Systems (ITS) as part of the overall security solution. As distinct from the intrusion avoidance of current systems, ITS systems focus on containing malicious faults and automatic recovery, which is similar to failover in fault-tolerant systems. Given this reality, while becoming an emerging software architecture paradigm, SOA does pose challenges to the design of intrusion tolerant systems as recognized in [Pal2009]. Although there has been work done in making services intrusion tolerant [Str2004, Ver2006], to establish a design framework based on mathematical and stochastic foundation has not been the focus.

1.2 Contribution

Motivated by the challenge of securing services in SOA, the contribution of this dissertation can be highlighted as follows:

a) Propose a taxonomy for intrusion tolerance architectures. The taxonomy is based on the main mechanism designed to provide resilience to systems, applications and services [Ng2010]. The three main categories identified are: intrusion-detection based, algorithm-based, and recovery-based.

b) Establish a systematic framework called Quantitative Framework for Intrusion Tolerant Services (QFITS) that can facilitate the end-to-end design and operation of intrusion-tolerant services. Since service consumers may have various needs in terms of

security and resilience levels, service providers should consider offering different levels of intrusion tolerance. The framework starts with Self-Cleansing Intrusion Tolerance (SCIT) [Ng2010] as the underlying intrusion tolerance mechanism, then is extended with the combination of other mechanisms such as intrusion detection and diversity. Component diversity in a recovery-based intrusion tolerance has been promoted as the solution for moving target systems, a new concept used to elude malicious attacks [Hua2011].

c) Introduce the concept of Intrusion Tolerance QoS (IT-QoS) for which I propose a specification based on UML (Unified Modeling Language), and adapted from the UML Profile for Fault Tolerant System [Obj2008]. For a service implemented as Web Service, a mechanism is required to allow IT-QoS registration and discovery.

d) Model the SCIT-protected services, including atomic and composite services that are essential to SOA, as Semi-Markov Process. This modeling method is adapted from the framework described in the seminal paper by Madan [Ma2004] tailored to services protected by SCIT, a recovery-based intrusion tolerance scheme, such that the solution of the steady-state probabilities of the Semi-Markov Processes leads to analytical expressions of the IT-QoS metrics such as Availability, Mean Time To Security Failure (MTTSF) , and Reliability.

e) Propose the mathematical analysis that allows the discovery of the correlation between the IT-QoS metrics and the parameters that characterize and control the associated intrusion tolerance system. This correlation would constitute the foundation for the

engineering approach during the design and runtime phases to ensure differentiated levels of certain IT-QoS metrics. Since the approach is supported by the quantitative analysis, the intrusion tolerance levels for services can be guaranteed with some degree of confidence. In my research, I have shown that such correlation exists in the cases when: a) simple SCIT is used; b) SCIT is combined with Intrusion Detection System (IDS); and c) SCIT replicas have diversity.

f) Propose a space-time strategy to maintain desired IT-QoS levels by inserting the service's attack surface metric into the analytical model. Attack surface has been studied in recent literature, as a measure of potential vulnerabilities of services [Man2011, Liu2008, How2002]. Using the analytical expressions obtained from the solution of a Semi-Markov Process established for SCIT-protected services, I have obtained mathematical results exhibiting how the tuning of the exposure window in SCIT can compensate the expansion of a service's attack surface, in order to achieve or maintain certain desired levels of IT-QoS.

1.3 Significance

Securing services in a Service-Oriented Architecture is a challenge to system designers, partly due to inherent software vulnerabilities, but also because of constantly evolving malicious attacks. The line of defense cannot solely rely on intrusion prevention and detection at the network periphery, and an authentication scheme, but must include intrusion tolerance at the service level. Thus, recognizing the need to make services intrusion tolerant to fulfill the requirements of operational security resilience of a system,

the goal of this research is to determine whether it is possible to establish a systematic and quantitative approach for modeling, designing and implementing differentiated Intrusion Tolerance QoS (IT-QoS) levels for applications and services in SOA. The approach has the following characteristics:

- It provides a quantitative basis for an end-to-end framework from modeling a service with IT-QoS parameters to realizing those QoS with necessary computing resources, and maintaining the desired QoS levels during the operational phase of the service.
- It enables the tuning of the IT-QoS to satisfy various needs of different service consumers, and to adapt the IT-QoS levels according to the dynamic runtime environment.

My research contribution is significant for the following reasons. First, QoS is important in a Service-Oriented environment. Indeed, service consumers want to select services based not only on their functionalities but also on their QoS values such as response time, throughput, and availability. In response to those requirements, service providers are motivated to provide services with published QoS levels that have to be guaranteed. Often, QoS precisely plays an important differentiating factor among Service Providers for the same functionalities. Moreover, during the design phase of an SOA system, software architects construct composite services and applications by utilizing atomic services or lower layers' services, whose aggregated capabilities and QoS satisfy the overall functional needs and quality constraints.

Second, security represents a crucial QoS, since today's applications and services are generally delivered over enterprise or even public wide area networks, which are susceptible to malicious attacks. With the understanding that Intrusion Tolerance is necessary to complete the defense-in-depth strategy for services, I propose that the notion of QoS be extended to Intrusion Tolerance component in order to ensure the resilience of a service against security faults. Furthermore, service providers should be able to offer different levels of intrusion tolerance in order to a) respond to various levels of security requirements of service consumers; and b) adapt to the dynamic operating environment from the security perspective.

Finally, a mathematically-driven approach would help software and system architects in the process of designing and implementing desired security QoS, thereby assuring some rigor and degree of confidence in guaranteeing published QoS levels. If a quantitative framework could be established, it could bring benefits in terms of system engineering activities such as hardware sizing and computing resource allocation, so that promised intrusion tolerance levels can be achieved. In the case of services delivered via Cloud Computing, this information might help engineers to validate the resource partition with the desired intrusion tolerance levels expressed in service level agreements.

1.4 Dissertation Organization

This dissertation comprises 10 chapters, including this Chapter 1. Chapter 2 introduces the taxonomy to classify intrusion tolerance architectures, provides a brief comparative discussion, and points out the control parameters of each architecture. Some of these control parameters will be utilized in subsequent chapters for quantitative analysis. In

Chapter 3, I describe QFITS, the proposed Quantitative Framework for Intrusion Tolerant Services, which is based on two elements: a) one is the SCIT architecture adapted to support services in an Service-Oriented Architecture; b) the notion of IT-QoS (Intrusion Tolerance QoS) providing a vehicle to systematically specify the intrusion tolerance quality of a service. Chapter 4 starts with modeling a simple service protected by SCIT using Semi-Markov Process, then obtains analytical expressions for the IT-QoS. The methodology to derive relations between the IT-QoS and the intrusion tolerance control parameters is laid out. Therefore, Chapter 4 serves as a basic foundation for subsequent analysis of various combinations of SCIT and a couple of other intrusion tolerance schemes. In Chapter 5, I extend the analysis performed in Chapter 4 to the case of a composite service. Based on the Semi-Markov Process, the combination of SCIT and IDS (Intrusion Detection System) is studied in Chapter 6, in order to obtain analytical results similar to the ones in Chapter 4. Chapter 7 revises the results obtained in Chapters 4 through 6, with the introduction of the attack surface of a service. Furthermore, building up from the work described previous chapters, I demonstrates how to leverage SCIT and the attack surface to achieve a space-time strategy for intrusion tolerant services. Chapter 8 can be considered as the epitome, because it describes the application of the probabilistic analysis to design an intrusion tolerant service. I describe C-SCIT, a cloud-based extension of the SCIT architecture in Chapter 9. Chapter 10 contains the conclusion and proposed some directions for future work.

1.5 Tools

Sagemath [Sag2014] has been used for plotting functions and computing numerical examples of the functions obtained and simulation data, while Wolfram Mathematica Online Integration [Wol2014] has helped finding the symbolic integral of an expression involving Weibull distribution.

CHAPTER 2 - INTRUSION TOLERANCE ARCHITECTURE TAXONOMY

2.1 Overview

Intrusion tolerant services are resilient to malicious attacks on networks, hardware and software which constitute the operating environments of these services. In the security context, service resilience can be characterized by the following qualities:

- Resistance to intrusions. The service is not "easy" to be compromised by malicious actors. This can be manifested by a prolonged mean time to failures.
- High availability. The service is still available to legitimate clients in the face of intrusions.
- Short recovery time. The service is quick to contain intrusions, recover to its clean state, and be available to legitimate clients.

In this chapter, I will survey the major intrusion tolerance mechanisms that exist in the literature. In my survey, I will perform a qualitative analysis of these various mechanisms in order to identify their control parameters, which can be later used as input into my framework to design services with intrusion tolerance QoS.

2.2 Taxonomy

On the surface, intrusion tolerance systems (ITS) seem to use the same mechanisms utilized in fault-tolerance systems such as fault detection, redundancy, and recovery functionalities. Although they have the same terminology, these mechanisms are actually

totally different in their design and realization. First, ITS utilize these functionalities to deal with malicious faults instead of system faults. Second, the differences originate from the different nature between failures due to system versus due to bad actors. For instance, fault-tolerant systems use redundant nodes to improve high availability. But, redundant nodes in an intrusion tolerant system may instead participate in a voting algorithm in order to detect compromised nodes and make services available to clientts.

In [Ng2010], a taxonomy based on whether intrusion detection is the main trigger for recovery to normal state of the service has been proposed. According to that taxonomy, we have four categories of ITS: detection-based, algorithm-based, recovery-based and hybrid. I will classify the existing ITS architectures into these four categories. During the classification, I will also determine their control parameters, some of which will be used in the case studies of my proposed framework.

2.3 Detection-based Architecture

Detection-based architectures rely on the detection capability to increase system survivability, in the sense that intrusion containment, system recovery and/or reconfiguration are triggered when the detection component can assert that there is an intrusion. In other words, detecting intrusion is a pre-requisite of any subsequent system's reaction to ensure tolerance.

With the proposed taxonomy, the following systems can be considered as Detection-based.

a. SITAR (Scalable Intrusion-Tolerant Architecture) [Wan2003a] aims at protecting services and applications, which are implemented by COTS products, against both known and unknown external attacks. The architecture consists of inserting SITAR components and connecting these components with existing COTS. Thus, the solution does not require COTS modifications. Basically, SITAR detects compromises and adaptively reconfigures the compromised servers. The detection occurs at two levels:

- The first level of detection is performed by a component doing the acceptance testing, based on criteria that may be specific to the application.
- The second level of detection is implemented by a group of servers joining a voting algorithm. These redundant servers have the same function, but are usually implemented with diversity in order to reduce the likelihood of failing under the same line of attack. With the voting algorithm, SITAR can produce the correct response, even in the case of an intrusion. However, the ability of masking intrusions is proportional to the number of redundant servers; N servers can mask (N/2 - 1) intrusions.

b. DPASA (Designing Protection and Adaptation into a Survivability Architecture) comprises multiple zones and layers with network-based intrusion detection systems to protect assets against malicious attacks [Pal2007]. Proxies are configured in the outer zone to intercept and inspect incoming traffic, and the host managing the security defense is located in the innermost zone.

c. The Willow architecture has intrusion detection sensors used to monitor application hosts that can be distributed over the wide area network [Kni2002]. What is

special with Willow architecture is its ability to analyze system vulnerabilities and detected faults, and to reconfigure in a distributed computing environment. This reconfiguration allows the architecture to achieve fault avoidance as well as fault tolerance.

d. Peng Liu has proposed ITDB (intrusion tolerance database) architectures [Liu2003]. ITDB focuses on how to contain intrusions, eliminate them, and repair corrupted data in the database. Triggering these actions is done by the detection layer. The effectiveness of intrusion isolation and data repair is conditioned by the speed of detecting intrusion.

e. DIT (Dependable Intrusion Tolerance) is composed of a cluster of mediating proxies coupled with a monitoring system for alerting the system when intrusions are detected [Val2002]. Like SITAR architecture, the cluster in DIT can detect anomalies based on an agreement protocol. At the high level, HACQIT (Hierarchical Adaptive Control of Quality of Service for Intrusion Tolerance) also uses an agreement protocol to augment its intrusion detection capability that is performed by regular IDS software agents [Jus2002].

f. While the previous architectures have application level intrusion detection capability, ITSI (Intrusion-Tolerant Server Infrastructure) focuses more on detecting intrusions at the network layer [Obr2003].

2.4 Algorithm-based Architecture

Algorithm-based intrusion tolerant systems utilize redundancy to implement algorithms such as voting algorithm, threshold cryptography, and fragmentation redundancy

scattering (FRS) to harden the resilience of the systems to be protected.

a. PASIS (Perpetually Available and Secure Information Systems) is a design to make storage systems survivable in the face of malicious intrusions [Gan2003]. A data object is encoded using a k-threshold secret-sharing scheme and dispersed in storage media, so that a hacker has to take hold of at least *k* secret pieces in order to assemble the data object. Thus, increasing intrusion tolerance would require increasing the number of replicas and threshold numbers.

b. MAFTIA (Malicious and Accidental Fault Tolerance for Internet Applications) is composed of layers of trusted services; a layer relies on trusted services in a lower layer to build more trusted services [Str2004]. At the end, MAFTIA provides to the applications a set of intrusion tolerant authority and transaction management services, that in turn are developed on a platform of common services. The latter services implement voting algorithms and *k*-threshold cryptography to be utilized by authority and transaction management services. MAFTIA architecture does incorporate IDS sensors; however, the main thrust of its architecture doesn't focus on IDS capability for protecting applications.

c. ITUA (Intrusion Tolerance by Unpredictable Adaptation) is another middleware that adaptively protects objects in applications [Cuk2001]. The design is also based on protocols for group communications and cryptography. ITUA adapts Cryptography techniques, challenge-response protocols, and *k*-security models are adapted by ITUA in order to handle the specific characteristics of a wireless sensor network.

2.5 Recovery-based Architecture

Recovery-based intrusion tolerant architectures are based on automatic restoration of the system to a known good state. This restoration can be periodic, and does not have to depend on the result of the intrusion detection component. In fact, some architectures in this category don't need to have intrusion detection to function.

a. SCIT (Self-Cleansing Intrusion Tolerance) architecture is simple [Hua2006]. At the core is the Controller responsible of managing a cluster of servers with identical services through a round-robin cleansing scheme to bring the cleansed server back to its pristine state. Those servers may be implemented or configured with some diversity in order to reduce the likelihood of being attacked by the same mechanism. The interface between the controller and the managed servers consists of a trusted connection. A server in the SCIT cluster continuously goes through three states: offline cleansing, online backup, and online primary. During the online primary state, a SCIT server is exposed to the Internet to serve incoming transactions; the duration of this state is called *exposure time*. In SCIT architecture, the cluster's servers can be implemented using virtualization technology.

b. Recovery-based intrusion tolerance has its analog in software rejuvenation. Khin Aung et al. have proposed software rejuvenation and redundancy to increase system survivability despite of the inevitability of software faults and aging [Aun2005]. The approach and the impact of the rejuvenation rate have been supported by the mathematical analysis of stochastic process modeling.

c. With respect to implementation, Hans Reiser and Rüdiger Kapitza have designed

proactive recovery of the system by using Hypervisor virtualization [Rei2007].

2.6 Hybrid Architecture

Besides the three major categories, there are intrusion tolerant architectures which actually combine various mechanisms contained in different categories.

a. For instance, combining periodic system rejuvenation with reactive recovery was proposed by Paulo Sousa et al. In their proposed architecture, the automatic rejuvenation complements the recovery triggered by a detected threat that exceeds a predefined threshold of tolerance [Sou2007].

b. The FOREVER [Sou2008] service combines recovery and diversity. Recovery can be triggered by a malicious event or follows a periodic reconfiguration like SCIT to proactively remove undetected malicious intrusions.

c. COCA (Cornell Online Certification Authority) is a certification authority which combines threshold cryptography and proactive secret-sharing [Zho2002].

d. As an option for achieving intrusion tolerance, Huang proposed a "moving target defense" using detection, recovery and diversity in the technology stack [Hua2011].

2.7 Discussion

The analysis of three examples of intrusion tolerance architecture, namely SITAR, MAFTIA, and SCIT have been discussed in detail in [Ng2010]. The summarized comparison is as followed:

- Detection. In the process of performing intrusion detection, SITAR may have to inspect the payload of the request in order to detect anomalies and signatures registered in its internal database. MAFTIA relies more on redundant replicas

endowed with voting algorithm for intrusion detection. SCIT has no detection, and provides intrusion tolerance by means of periodic and deterministic cleansing.

- Data Protection. Thanks to the continuous cleansing of the service, SCIT can limit the loss of data. This can be achieved by limiting the exposure window with which a service is prone to attacks. For other intrusion tolerance schemes, data protection depends on the success rate of the detection.
- Recovery. Except for SCIT, recovery and reconfiguration of the service and components of the service are triggered only when an intrusion is detected. The detection can be performed by intrusion detection sensors or via various kinds of testing. SITAR has an array of testing that aims in detecting the occurrence of malicious compromises. Besides testing, SITAR also utilizes a voting algorithm to detect and mask intrusions as MAFTIA.
- Performance. Unlike SITAR and MAFTIA, SCIT executes its intrusion tolerance scheme out of band, the performance of the protected service is not affected.
- Integration. Both SITAR and SCIT can be easily configured to protect a service. On the contrary, MAFTIA provides a platform and middleware on top of which a service can be developed and/or integrated to utilize the intrusion tolerance features.

2.8 Control Parameters

An interesting question is what the control parameters of an intrusion tolerance architecture are.

Definition 2.1. An intrusion tolerance control parameter, denoted by ITS-P will have the following characteristics:

a) ITS-P is quantifiable;
b) The variability of ITS-P impacts the variability of the service's intrusion tolerance.

For detection-based intrusion tolerant system, the control parameters are the probability of detection and probability of false positive [Nag2011].

For recovery-based architectures, Exposure Window is the only control parameter. Server node redundancy is used in SCIT architecture. However, the number of replica nodes depends on the exposure window. Hence, exposure window is the control parameter for recovery-based architectures.

Most of the algorithm-based intrusion tolerant systems rely on key sharing, voting algorithm, etc., so the control parameter can be the redundancy number k.

If diversity is used, then the control parameter is V, the number of diverse versions of a service. Traditionally, N-version programming has been used to increase system fault tolerance. According to Huang [Hua2011], one can obtain variants of a service by considering layers on top of which the service is deployed. Such layers are, for instance, operating system, application server, presentation layer. As a result, integrating different versions of multiple layers to configure and deploy a service would give a combinatorial number of versions of a service.

In a later chapter, I will show how these control parameters impact the QoS of a service protected by intrusion tolerant mechanisms.

CHAPTER 3 - DESIGN FRAMEWORK QFITS

3.1 Overview

The purpose is to provide to system and software architects an end-to-end framework to design, configure and deploy services such that they satisfy a set of intrusion tolerance QoS. This thesis focuses on the quantitative design methods to make a service resilient to malicious attacks; the interatction with other QoS is not considered, and can be another topic of research. Given the motivation of a quantitative approach for designing resilient services, I propose the Quantitative Framework for Intrusion Tolerant Services (QFITS), which includes both the computational and modeling elements. At the foundation, the intrusion tolerance nature of QFITS is based on the recovery-based intrusion tolerance mechanism of Self-Cleansing Intrusion Tolerance (SCIT). Although starting with SCIT as the foundation, I expect the framework to be extensible to other intrusion tolerance mechanisms which can work in tandem with SCIT. As an illustration, I will study the framework as applied to an architecture having an intrusion detection component supplementing the SCIT mechanism.

3.2 Framework Overview

The framework QFITS comprises four major components as depicted in Figure 3.1. Let:

- $P = \{p_1, \ldots, p_n\}$ be the set of parameters that control the behavior of an ITS; and

- M = $\{m_1, \ldots, m_q\}$ the set of IT-QoS metrics of interest.

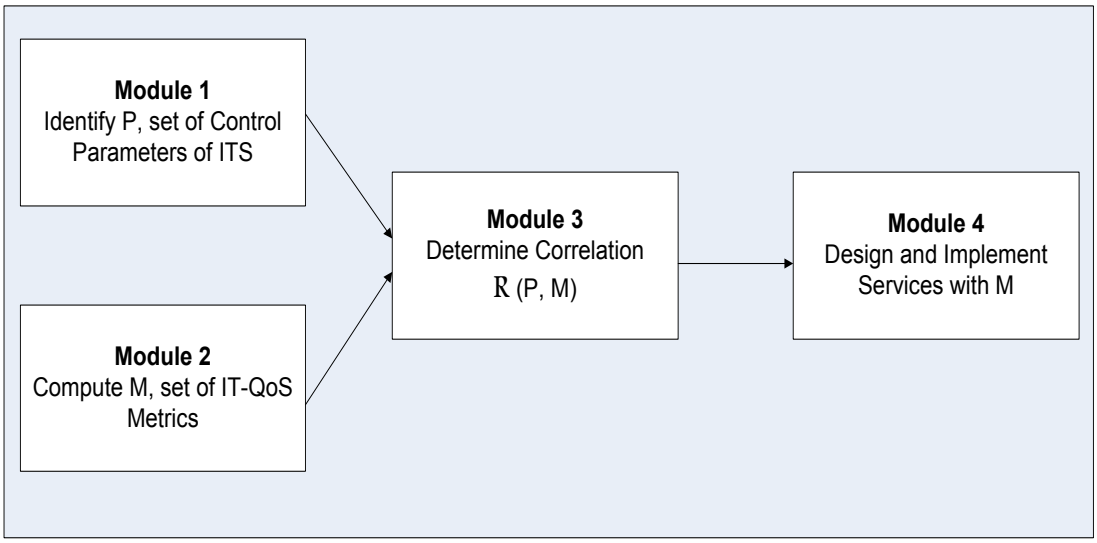

Figure 3.1. QFITS Modules

For Self-Cleansing Intrusion Tolerance (SCIT), the set P is defined as P = {W}, where W is the exposure window. For a detection-triggered ITS, the set P may comprise of the detection probability P_D and the false-positive probability P_{FP}: P = { P_D, P_{FP}}. For a voting algorithm based ITS as mentioned above, we can have P = {k}, where k is the number of required participants of the intrusion tolerance scheme. Mean Time to Security Failure (MTTSF) could be considered to be included in the set M in the analysis of QFITS.

The framework QFITS consists of four modules as followed:

- Module 1: Parameter Identification.

Module 1 identifies the set P of parameters that control the behavior of the ITS mechanism. Usually, the elements in P are closely associated with the scheme employed in the intrusion tolerance solution, and can be determined via a qualitative analysis of the ITS scheme. However, the validation of the set P can be performed implicitly in Module 3 described below.

- Module 2: QoS Computation.

This Module computes the set M of IT-QoS metrics, using Semi-Markov Modeling of the ITS. According to [Ma2004], this step would require modeling the states and transitions of the system in the face of facing malicious attacks and protected by intrusion tolerant mechanisms, and selecting of QoS metrics of interest for M. For instance, one would be interested in M = {Availability, MTTSF}. Then, the elements of M could be defined in terms of limiting probabilities of the Semi-Markov Chain used to model the ITS being studied.

- Module 3: Correlation Discovery.

This Module follows the previous one, and represents the key step in my research proposal. From the set of parameters P determined in Module 1 and the probabilities obtained in Module 2, a correlation between M and P has to be inferred. This correlation is embodied as an algebraic expression of M in terms of P, or an expression of P that bounds the elements in M.

Definition 3.1. M is correlated to P denoted by $\Re$ (P, M) if there exist multivariate functions $F_1, \ldots, F_q$ of the parameters in P, and binary relations $R_1, \ldots, R_q$ such that:

$$\begin{cases} m_1 \; R_1 F_1(p_1, \dots, p_n) \\ \quad \dots \\ \quad \dots \\ m_q \; R_q \; F_q(p_1, \dots, p_n) \end{cases}, \qquad (3.1)$$

R_1, …, R_q are either equalities or inequalities. In the case where closed form expressions cannot be obtained via symbolic manipulation, then statistical analysis of experimental measurements could be used to arrive to the sought-for correlation.

With this definition, the problem of Module 3 can be formulated as the existence of the relation $\Re$ between the two sets P and M.

- Module 4: QoS Modeling and Design.

The correlation obtained in Step 3 serves as the basis for designing different levels of QoS metrics in the set M for the services.

In QFITS, the major steps are the computation of the metrics in M and the establishment of the relation between P and M, since they enable the realization of developing services with varying levels of IT-QoS. While computing IT-QoS metrics in M can be performed based on Semi-Markov Chain models, the discovery of a mathematical relation between P and M is quite challenging.

3.3 Examples

Following is a purely fictitious example. Assume that a system architect wants to design an intrusion tolerant service with high availability using SCIT. Then:

- For determining the set of control parameters in Module 1, he has P = {W}, with W being the SCIT exposure window.

- For Module 2, the set M = {SA}, with SA being the secure availability of the service. Using Semi-Markov model, he has to compute SA in terms of steady-state probabilities of the model. Let P_0, …, P_n be the steady-state probabilities. He can obtain an algebraic expression for SA. For the sake of the argument, let's say he found that SA is inversely proportional to the probability of attack P_A:

$$SA = \frac{k_1}{P_A} \qquad (3.2)$$

- For Module 3, he has to find the correlation between W and SA by studying the correlation of (P_0, …, P_n) and W. For example, he found that P_A is proportional with W:

$$P_A = k_2 W \qquad (3.3)$$

Then, he can find a correlation between A and W:

$$SA = \frac{k_1}{k_2 W} \qquad (3.4)$$

Instead of the equality, if (3.3) is an inequality such as:

$$P_A \leq k_2 W \qquad (3.5),$$

then, (3.6) will become a lower bound for SA:

$$SA \geq \frac{k_1}{k_2 W} \qquad (3.6).$$

From (3.4) or (3.6), he can deduce that SA will increase if the exposure window decreases.

- For Module 4, he will use the results from Module 3 to design the service using UML language, as availability can be tuned by varying W.

3.4. Service-Oriented Architecture

A Service-Oriented Architecture (SOA) is based on loosely coupled services conceivably arranged in layers, from physical layer, core service layer, business logic, and presentation layer. These services can be atomic, or composite services, which are themselves composed of atomic services or even other composite services. There exist mechanisms, for instance via a process language such as BPEL (Business Process Execution Language), that orchestrates services together to construct more complex services. Service is the building block of the software system in the SOA environment. A Service is designed, implemented, deployed, and published with its defined interfaces in a Service Registry. Thanks to the registry, a Service Consumer can discover the needed services, and invoke the service operations according to its published specifications, including communication mechanism, parameters required to execute requests, return and error codes, and schema of data to be exchanged. A service provider can publish its web service in a UDDI (Universal Description, Discovery and Integration) [Oas2004] registry using the WSDL standard. Not only service's functionalities should be published, but they should be accompanied by non-functionality requirements or QoS characteristics, such as reliability, availability, and response time [Dam2006]. In this case, Service consumers will have the capability to select services that satisfy both the desired functionalities and QoS constraints. In addition, software architects will be able to design applications or composite services from component services such that: a) the composition contains all desired functionalities; b) the aggregated QoS values of the component services satisfy the QoS constraint of the composition. Thus, the questions are:

- What mechanism a service designer on the provider side can use to publish QoS levels of the designed services?
- What methodology a service designer can use to design a service with desired QoS constraints?

3.5. QoS Quantification

QoS is a non-functional requirement, and quantifying a QoS is a challenge, especially security-related QoS. There has been a lot of active research done in the area of QoS for SOA services [Irv2001][Boc2008]. Instead, my proposal wants to look into the specific question of how to quantify security QoS for services protected by ITS. Then, I would like to introduce the notion of IT-QoS (Intrusion Tolerant QoS). IT-QoS is a subset of the security-related QoS metrics of particular interest, following the above discussion. Starting with SCIT, a recovery-based intrusion tolerance mechanism, I believe that the approach detailed by the QFITS framework will contribute to answering these questions, thus providing a systematic way for service designers and operators to achieve and sustain IT-QoS for services.

Menasce et al. have proposed a self-architecting framework named SASSY [Men2010]. The goal of SASSY is to provide an optimal selection of a sequence or composition of architectural patterns that satisfy the overall desired QoS value. Thanks to the Utility Function which offers a mathematical expression for QoS metrics, and heuristics algorithm, this pattern selection is feasible within the constrained timeframe and conditions so that the system architecture can adapt to different scenarios. If an intrusion tolerance component is considered as a QoS architectural pattern, then the output of

QFITS can serve as an input to SASSY in its process of computing the utility function and selecting the appropriate level of IT-QoS to satisfy the overall QoS.

In terms of quantitative security metric, Manadhata and Wing's work is based on the concept of Attack Surface determined by the set of resources in a software system that can become potential vulnerabilities [Man2011]. Thus, the security level of a system can be quantified by the extent of its Attack Surface: the larger the Attack Surface is, the less secure the system. The elegance of this concept lies in its I/O automaton model, so that all the formalism of automaton theory can be leveraged. However, the quantitative nature of the approach relies on the numerical assignments of damage-effort ratio of the resources - methods, channels and data - which could only be performed by experts or software developers.

A similar approach to Manadhata's is Schechter's econometric model, based on security risk, cost of security breach, and effort/penalty of attackers trying to compromise the system [Sch2005].

Instead of relying on subjective assignments during the computation process, my proposal is based on an end-to-end quantitative approach to IT-QoS design, utilizing stochastic process theory, and mathematical derivation. If this approach were realizable, then SOA architects would be able to design applications and services with desired IT-QoS levels. At the same time, system architects can perform sizing exercise to allocate adequate computing resources for ensuring required IT-QoS levels.

3.6 QoS Algorithm

Not only static configuration of services with specified QoS has been proposed, but dynamic and adaptive algorithms have also been designed and reported by Cardellini [Car2009]. These research works treat the QoS issue for SOA either at the generic level, focus on traditional system qualities, finding optimal solution for QoS selection, or optimizing the selection algorithms.

3.7 Security QoS

Peng Liu introduced the notion of QoIA (Quality of Information Assurance) in the context of intrusion tolerant database [Liu2003]. Add-on components to an intrusion tolerant database system help addressing different levels of service quality: on one hand, clients can register their required QoIA levels; on the other hand, service providers will be able to correct problems caused by malicious intrusion so that desired levels of QoIA are guaranteed. Internally, the intrusion tolerant components of the architecture execute special algorithms in a database management context to contain the spread of the attacks and rollback corrupted data back to their original state, while still serving current transactions. Instead of modifying a database COTS (Commercial Off the Shelf) product, those intrusion tolerant components interface with the database management server to achieve intrusion resilience. However, as far as I am aware, no parameters were defined for the concept of QoIA; in addition, QoIA was mentioned in the context of transaction processing by resilient databases.

In the domain of security related QoS, the authors of [Irv2001] discussed the usefulness and efficiency of Quality of Security Service (QoSS), and introduced the so-called

"Security range", instead of binary values. Security range can be realized by making a trade-off study between the desired level and the cost incurred by computing resource usage. In this regard, the scheme and architecture proposed in QFITS can be considered as an enabler for providing IT-QoS ranges.

3.8 Fault Tolerance QoS

UML extensions for QoS and Fault Tolerance are provided in [Obj2008], which specifies the meta-model with four entities:

- QoS Category is the top entity used to group multiple QoS Characteristics of the same area, such as Performance, Dependability, and Security.
- QoS Characteristic is a specific non-functional aspect. For instance, Throughput and Latency are two Characteristics of Performance Category.
- QoS Dimensions are used to quantify a QoS Characteristic. Network latency and jitter delay are two QoS Dimensions of the Latency Characteristic.
- QoS Constraint, as the name says, specifies the constraints and boundary values of QoS Dimensions.

3.9 IT-QoS

Inspired by the work done in [Dam2006, Obj2008], I propose to extend the QoS concept to intrusion tolerance. Consequently, IT-QoS characteristics can be modeled during the design of services, as with other traditional QoS. Since the notion of *fault* is extended to include malicious fault caused by malicious actors aiming at breaking the security of the services [Str2004], there is an analogy between service fault tolerance and intrusion tolerance, thanks to which, it would be natural to adapt fault tolerance QoS to the domain

of intrusion tolerance. The following table shows the parallel between the fault tolerance QoS excerpted from [Obj2008] and the QoS Characteristics for intrusion tolerance.

Table 3.3.1. IT-QoS Characteristics

Fault Tolerance		Intrusion Tolerance	
Fault Tolerance Algorithm	Schemes, algorithms, and mechanisms used in the architecture to detect and correct system faults.	**Intrusion Tolerance Algorithm**	Schemes, algorithms, and mechanisms used in the architecture to detect, contain, and correct malicious faults.
Reliability	Metric showing the service ability to operate correctly for a time interval.	**S-Reliability**	Metric showing the service ability to operate correctly and unaffected by malicious faults for a time interval.
Availability	Probability measuring the service ability to operate.	**S-Availability**	Probability measuring the service ability to operate without being affected by malicious faults.
Maturity	Metric of the service to avoid system faults.	**S-Resilience**	**MTTSF** Mean time to security failure: time between two security compromises of the service.
Recoverability	Metric showing the ability of the service to recover from system faults and restart.		**MTTSR** Mean Time to Security Recovery, metric showing the ability of the service to recover from malicious faults.

3.9.1 Intrusion Tolerance Characteristic

Definition 3.1. Intrusion Tolerance Characteristic specifies the features or QoS Dimensions of the intrusion tolerance architecture.

Those dimensions are:

- *maximum-intrusion-number* (maximum threshold of attacks that the ITS can survive). Examples can be found in MAFTIA middleware such as Transaction Manager, Authentication Manager. ITS with redundancy and various voting algorithms make the protected services stand in the face of a maximum of number of intrusions. Similarly, COCA, which is based on a shredding algorithm of security certificate has an upper tolerance limit called threshold, that an attacker has to overcome in order to compromise the certificate. Analogous principle of information shredding among participants is also used for intrusion tolerant storage systems.
- *intrusion-processing* (mechanism to detect and cleanup the intrusion). ITDB [Liu2003] contains an elaborate algorithm to detect intrusions to database systems and isolates attacks to the data in the database. The challenge depends on the transaction rate, which influences the propagation rate of the data corruption, and the latency of the detection. The longer it takes to detect an intrusion, the more complex and time-consuming it takes to isolate and rollback the data corruption.

- *intrusion-treatment* (mechanism to recover from attacks). All ITS have a recovery scheme to restore compromised services and components to their clean state. The difference is what triggers the recovery. ITS recovery is automatically initiated, mostly followed the assertion of an intrusion by the detection component. For others such as SCIT, recovery is periodic or pro-active based on pre-configured parameters [Hua2006, Sou2007].

3.9.2 S-Reliability

S-Reliability extends the notion of service reliability in the SOA context, where reliability is defined as the probability that a service operation performs correctly without failure [Lim2001][Pha2006].

Definition 3.2. Service S-Reliability is the probability that a service performs reliably and correctly in an environment that is unaffected by malicious intrusions.

Thus, S-reliability Dimension is the value of that probability for a time period, say one year of operation for instance. Note that [Obj2008] uses the dimension "expected-number-service-failures".

3.9.3 S-Availability

Definition 3.3. S-Availability specifies the probability that the service is operational without being affected by malicious faults.

One Dimension of S-Availability is the Availability Value.

3.9.4 S-Resilience

Definition 3.4. S-Resilience specifies the ability of the service to operate in the presence of malicious faults.

The Dimensions of S-Resilience are:

- Mean time to security failure (MTTSF), which is the time between two security compromises of the service;
- Mean time to security recovery, which is the time to reconfigure and recover from malicious intrusions (MTTSR),

Note that the above table shows that S-Resilience corresponds to two QoS Characteristics from Fault Tolerance, namely Maturity and Recoverability.

3.10 Modeling IT-QoS

Given the above IT-QoS, a software architect can model a system and service design with desired values of QoS, using UML annotations and stereotypes as described in the Schedulability, Performance and Time Specification (SPT) framework [Uml2005]. The following diagram taken from [Ng2010] shows an example of a case study of the access component of a digital record archival system. Digital objects are stored in a Trusted Digital Repository fronted by the service StoreDigitalRecords. The service AccessDigitalRecords validates the roles and responsibilities of the Consumer who requests to access digital objects via the portal AccessPortal. The collaboration diagram shows the modeling of the DigitalRecordStorage and AccessDigitalRecord services with desired S-Reliability values.

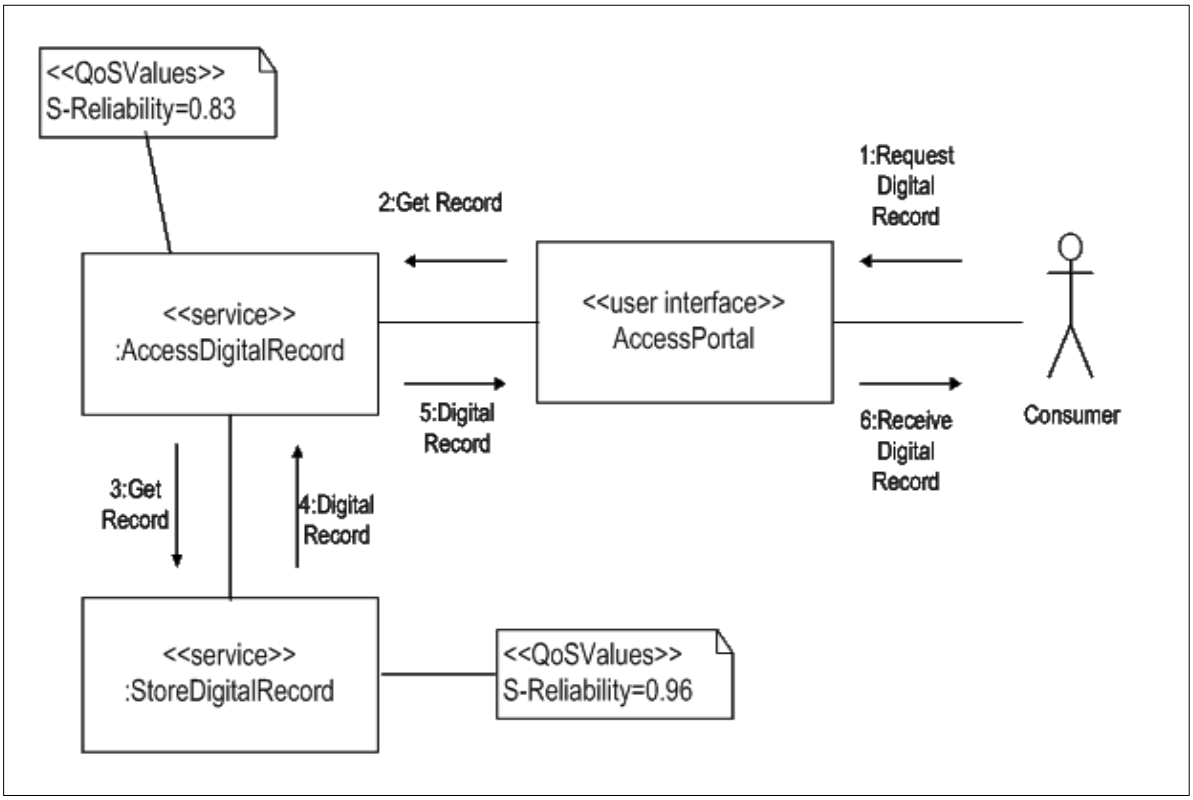

Figure 3.2. Collaboration Diagram with IT-QoS values [Ng2010]

3.11 Publishing IT-QoS

For a service to be used by potential clients, its operations along with operation parameters and communication protocols, need to be published in a service registry. General QoS and IT-QoS should be also be publicized so that clients can select the services with desired QoS values. One possibility is to adopt the extension Q-WSDL proposed in [Dam2006] by extending it to intrusion tolerant Web Service. Continuing the case study above, the following snippet illustrates a specification of the IT-QoS elements

of the AccessDigitalRecord service enclosed in the element tag <IT-QoS> within the WS-Policy [Web2007] section within the WSDL.

Table 3.3.2. WSDL Snippet with IT-QoS for WS-Policy

```
<wsdl:definitions targetNamespace="http://example.org/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">

<wsdl:portType name="GetRecordPortType">
  <wsdl:operation name="GetRecord">
    <wsdl:input message="GetRecordInput"/>
    <wsdl:output message="GetRecordOutput"/>
  </wsdl:operation>
</wsdl:portType>
<binding name="AccessDigitalRecordBinding"
       type="tns:GetRecordPortType">
    …
</binding>
<wsdl:service name="AccessDigitalRecord">
<wsdl:port name="GetRecordPort"
       binding="tns:AccessDigitalRecordBinding">
  <soap:address location=http://example.org/accessdigrecord/>
</wsdl:port>

<wsp:Policy>
  <IT-QoS>
    <S-Reliability>
      <reliability-value="0.83"/>
    </S- Reliability>
  </IT-QoS>
</wsp:Policy>

</wsdl:service>

</wsdl:definitions>
```

3.12 SCIT Architecture

At the core, QFITS is based on the SCIT architecture [Hua2006], which consists of a Central Controller and SCIT nodes as depicted in Figure 3.2. The nodes can be virtualized machines hosting the services to be protected. Diversity can be applied to the

nodes to increase the intrusion tolerance by reducing the effectiveness of compromise attempts [Hua2011]. The nodes are usually grouped into clusters, each of which contains servers with identical functionalities, but with some diversity. For example, we can have a group of web servers, one running on Linux, another on Windows, and a third one on Mac OS. The benefit of diversity in operating system as in this example is to reduce the effectiveness of intrusion attempts.

The Controller can be deployed on a physical host separated from the one(s) used to deploy the nodes. Communication links between the Controller and the nodes are secure and trusted such that [Hua2006]:

- The link from the Controller to the online node is one-way so that the Controller is not on the data path that can be reached by any potential hacker coming via the online node.
- The link from the Controller to the offline nodes can be two-way. Note that offline nodes are not receiving any external traffic.

SCIT architecture is recovery-driven, i.e. the recovery of the servers, applications, and services to be protected by SCIT will go through automatic and periodic recovery. Figure 3.3 depicts the components of SCIT, which consist of the Central Controller and the SCIT nodes. The Central Controller manages and controls all the nodes to be protected. Each node within a cluster of similar nodes is directed by the Controller to constantly go through the following orderly lifecycle. The Controller executes an internal algorithm to direct each node in the group going through the continuous cycle of states. For services in an SOA system, a service can have multiple instances, each deployed on a SCIT node.

Since an instance of a service is contained in a node which undergoes periodic cleansing as directed by the Controller, the cycle of a service instance is the same as its hosting node:

- Live Spare state, where the service is in a known good state, but still offline;
- Active state of duration W_o, during which the service becomes online to receive and process incoming requests,
- Grace Period state of duration W_{GP}, when the service stops accepting new transaction requests and tries to complete outstanding requests still in its queue,
- Inactive state, where the service is offline and undergoes the restoration cleansing to a known good state.

The active state period W_o which is called SCIT exposure window, depends on the cleansing time specific to an application or service. In Figure 3.2, *Node 1* is in Active state, *Node 2* in Live Spare state, and *Node n* in Inactive state.

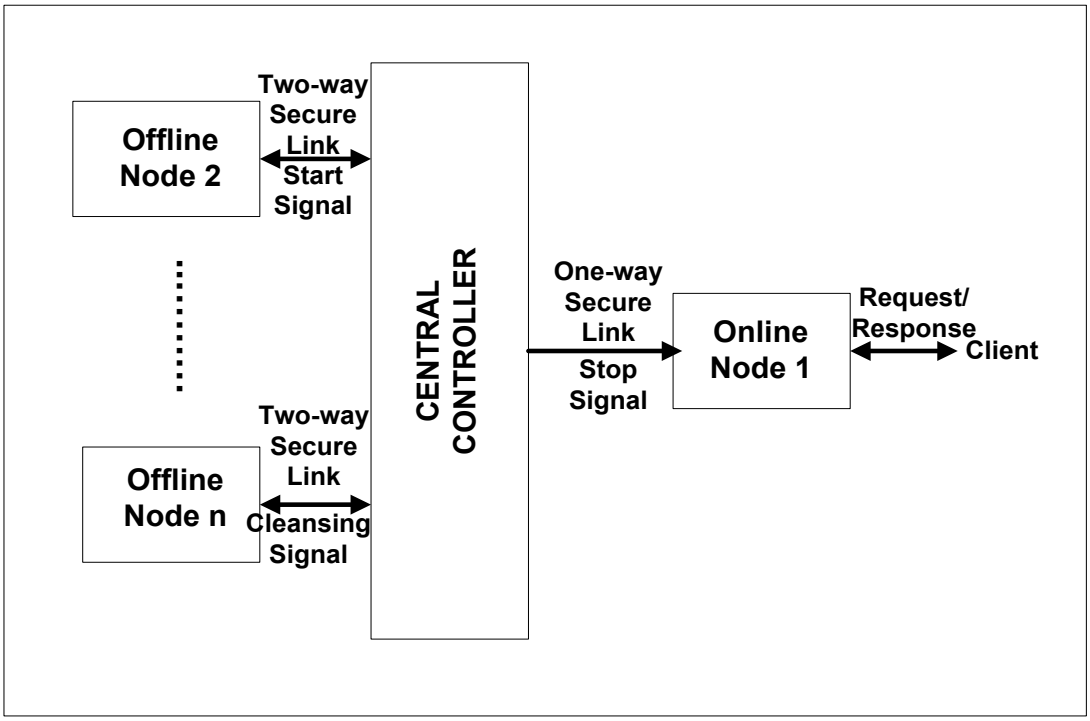

Figure 3.3. SCIT Architecture Components

Thanks to the clustering configuration and the continuous cleansing of the nodes, a service hosted by these nodes protected by SCIT exhibits the following characteristics:

- At any time, there exists an online service instance to serve incoming requests. The rotation algorithm executed by the Controller is responsible to ensure that there is no gap in terms of service availability during the swapping of service instances.
- A spare instance of the service is ready to be swapped in to replace the online instance, especially before the end of the online period of the live instance.

The cleansing of the nodes hosting the service instances is performed in the background, and transparent to the clients of the service.

The time period of the rotation through the four states above depends on the cleansing time, which is specific to an application or service. In Figure 3.2, *Node 1* is in Active state, *Node 2* in Live Spare state, and *Node n* in Inactive state.

3.13 SOA and Service Container

The current implementation of SCIT is based on virtualization technology such as VMWare, which allows the instantiation of multiple servers with diverse guest operating systems on a single host machine, and speed up considerably the rotation time. Thus, a server or node in the SCIT architecture is realized by a virtual machine running on a hardware platform with a specific configuration. Furthermore, a node becomes a container for the services sharing the same IT-QoS level requirements (see Figure 3.4).

Let (L_i), i=1..K, be K different levels of an IT-QoS to be provided. For each level L_i, we need N_i replicas for SCIT round-robin cleansing. Each of these N_i replicas contains exactly the same services sharing the same IT-QoS level L_i. In practice, a Service Container may contain an application server where services are deployed and activated.

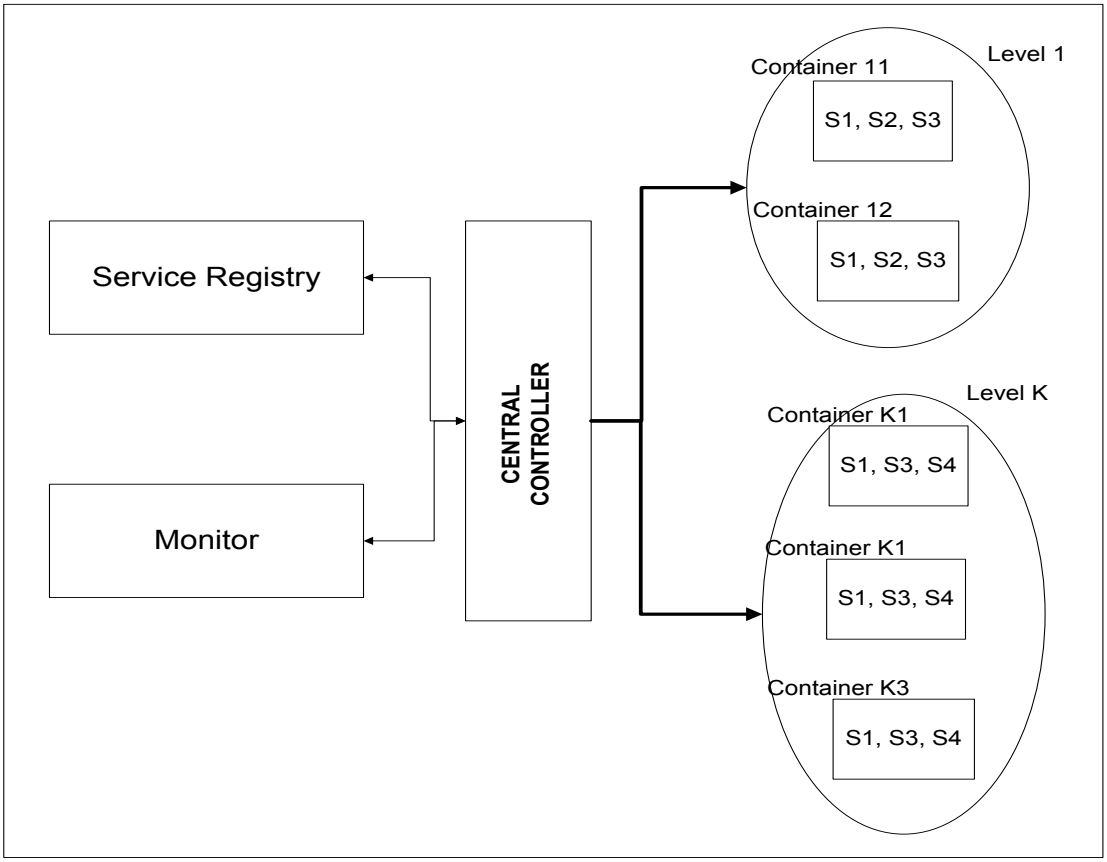

Figure 3.4. SCIT Service Containers

As depicted in Figure 3.4, the Central Controller interfaces with:

- The Service Registry in order to retrieve current IT-QoS characteristics of the services and publish updated IT-QoS values in the dynamic mode.
- The Monitor that allows the adjustment of the operating environment parameters. In a subsequent section, we will discuss more about the role of the Monitor in the system.

The Central Controller manages the containers belonging to all levels from Level 1 to Level K. Each group of containers corresponding to a level is represented by a dotted line. The diagram shows that Level 1 has a group of similar containers, from *Container 11* to *Container* $1N_1$, which have the same services S_{11}, S_{12}, etc. These services will ensure the same level of S-Reliability. Similarly, the group associated to Level K has the containers from *K1* to KN_K.

Moreover, two modules implementing the static model and the adaptive scheme will reside inside the Central Controller.

3.14 SCIT Parameters

The following table lists the parameters that are used to configure the SCIT architecture.

Table 3.3.3. SCIT Parameters

Notation	Parameter Name	Notes
W	Exposure window	$W = W_O + W_{GP}$ Total Time that a SCIT Node is active and processes transaction requests.
W_O	Online window	Time that a SCIT Node is active and processes transaction requests.
W_{GP}	Grace period	Time that a SCIT Node is online to receive transaction requests.
T_C	Cleansing time	Time needed to cleanse and reload a SCIT Node.
N	Number of SCIT nodes in the cluster	Number of SCIT Nodes that are needed to provide a guaranteed exposure window.

3.14.1 Grace Period Control

In the lifecycle of a service instance, the exposure window comprises of the online time W_O and the grace period time W_G :

$$W = W_O + W_{GP}.$$

Since the online service instance is open to accept data traffic and incoming service requests during the online window W_O, all attacks can only possibly arrive during that period. At the end of the online window, the Central Controller signals to the online instance to enter its grace period, and the spare instance to become online, so that the service has an online instance at all times. After W_{GP}, the service instance in grace period

enters the cleansing mode. In a subsequent chapter, I will establish the relationship between the exposure window and the desired IT-QoS parameters. Given that the grace period is intended to allow the service instance to finish the queued requests, I will compute an estimate of the grace period W_{GP} based on the total exposure window.

Let:

- N_R be the average number of outstanding requests in the queue when the server enters the grace period.
- Let μ be the service rate in terms of number of serviced requests per unit time.

If I model the incoming traffic and transaction requests by using a Poisson process with parameter λ, then the average number of outstanding requests cannot exceed the average number of arrivals during the online window:

$$N_R \leq \lambda W_O.$$

Then, an estimate of the grace period can be given by the time to serve the average number of outstanding requests:

$$W_{GP} = \frac{N_R}{\mu} \leq \frac{\lambda W_O}{\mu}.$$

In order for the system not to be overloaded, but adequate to serve the rate of incoming traffic, it is realistic to assume that:

$$\frac{\lambda}{\mu} < 1.$$

Then, I can obtain an upper bound for W_{GP}:

$$W_{GP} < W_O.$$

3.14.2 Number of SCIT Nodes

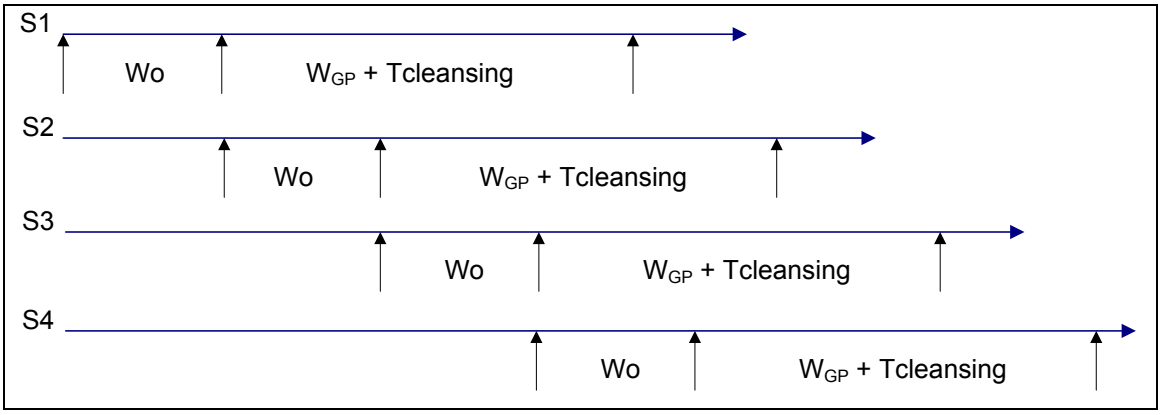

Figure 3.5. Timeline of SCIT events

Proposition 3.1. Let T_C be the cleansing time of a service, and W be the total exposure window. Then, SCIT can be designed such that the number of SCIT nodes N is given by:

$$N = \left\lceil \frac{2T_C}{W} \right\rceil + 2 \qquad (3.7).$$

Intuitively, we can make two observations:

• The smaller W_o is, the faster is the rotation cycle. Hence more nodes will be required.

• The cleansing time T_C has an adverse effect on the rotation cycle, i.e. the longer it takes to restore a node, the longer is the rotation cycle. Hence, in order to keep the same cycle period, we need more nodes. Based on this, we arrive at the expression for the number of nodes N that is needed to provide an exposure window of value Wo:

$$N = \left\lceil \frac{W_{GP} + T_C}{W_O} \right\rceil + 1 \qquad (3.8)$$

The extra node in the above expression is for the online node in the cluster. If we let W_{GP} = W_o, or W = W_{GP} + W_O = 2W_O, then we obtain the formula for the number of SCIT nodes as expressed in Proposition 3.1, that is:

$$N = \left\lceil \frac{2T_C}{W} \right\rceil + 2 \, .$$

For example, if T_C for a server takes 10 min, and we want W_o = 3 min in order to achieve a certain level for MTTSF, then the cluster requires N = 6 nodes. Whether N = 6 is feasible depends on the hardware configuration of the physical host. Indeed, as each node is implemented using virtualization, there is a hardware limit on the number of virtual machines (VM) that can be hosted. For this, we define N_{max} as the maximum number of VM(s) of a given physical configuration.

Numerical Examples. In the following examples, I let W_{GP} = W_O so that equation (3.7) can be applied. The goal is to derive the number of nodes, given the values for W_O and T_C.

Table 3.3.4. Example - Computing the Number of Nodes

T_C (min)	W_O (min)	N
10	100	3
10	10	3
10	1	12
100	100	3
100	10	12
100	1	102

Given these examples, it is clear that one of the considerations in the design of protecting a service by SCIT is to minimize the cleansing time. From the engineering point of view, it has been proven that the virtualization technology can facilitate this goal [Ban2009]. However, the design may become challenging if the service involves data whose volume can grow over time; certainly, the growth rate will be an additional factor. In particular, a closer look into the challenge posed by using SCIT to protect services generating "Big Data" with Volume and Velocity would be interesting.

CHAPTER 4 - SIMPLE SCIT AND ATOMIC SERVICE

4.1 Overview

In an SOA environment, a service can be atomic or composite. Usually, atomic services reside in the infrastructure layer, and constitute building blocks for the SOA ecosystem. In this chapter, I will describe how the SCIT architecture can be used to protect an atomic service. From this setting, I proceed to analyze the behavior of the service from the perspectives of intrusion tolerance using Semi-Markov modeling. The computation of the steady-state probabilities the Semi-Markov chain as applied to the atomic service protected by SCIT allows me to derive:

- The expressions of a set of IT-QoS: MTTSF, MTTSR, S-Availability, and S-Reliability (Module 2 of QFITS);
- The relationships between these IT-QoS and the SCIT control parameters, namely the exposure window, the grace period and number of SCIT nodes (Module 3 of QFITS).

From these relationships, SCIT algorithms performing the rotational cleansing of the service instances can be devised to satisfy the required IT-QoS levels. Modeling a generic intrusion tolerance system using Semi-Markov Process, and defining steady-state availability and MTTSF have been described by Madan et al. in [Ma2004]. In my research, not only was the Semi-Markov Process based methodology adapted to the case

of a service protected by SCIT, but the focus is to discover the relationship between the control parameters of the intrusion tolerant architecture and the IT-QoS metrics. This relationship is at the core of the QFITS framework, that will enable system architects to design and tune services with published IT-QoS. The idea of deriving the relationship between the SCIT exposure window W on one side, and MTTSF and S-Availability on the other side was first proposed in [Ng2009].

4.2 Semi-Markov Chain

Let E be the discrete state space, Ω the probability space, and X_n: Ω → E the random variable. Assume that X_n is a discrete-time Markov chain (DTMC), i.e. X_n satisfies the memory-less property [Ros2009]:

$$P_{ij} = (X_{n+1} = j \mid X_n = i, X_{n-1}, X_1, X_0) = P(X_{n+1} = j \mid X_n = i)$$

All values of P_{ij} form the transition probability matrix **P**. Thus, the DTMC X_n can be said to be defined by the state space E and the transition probability matrix **P**:

$X_n = (E, \boldsymbol{P})$.

Let T_n be the time when X_n changes from one state *i* to state *j* in S. Then, Y(t) = X_n for every *t* in [T_n, T_{n+1}) is a Semi-Markov chain, with X_n being the embedded DTMC. The interval length $H_n = (T_{n+1} - T_n)$ is called the sojourn time of the state of X_n at time T_n. So, a Semi-Markov Chain Y(t) can be defined by X_n and H_n:

$$Y = (E, X_n, H_n)$$

In the particular case where all sojourn times of all states follow exponential functions, Y(t) becomes a continuous-time Markov chain. Moreover, if the Markov chain X_n is ergodic, we can compute the steady-state probabilities of X_n.

4.3 Transition Diagram

In this section, the presented model an atomic service S is not for the internal states (Live Spare, Active, Grace Period, and Cleansing) of the service instances within the SCIT architecture. Actually, the Semi-Markov Process (SMP) represents service S as a whole as it reacts to malicious attacks. SMP is used since the sojourn time in each state of the process depends on the behavior of the attacker. Indeed, an attacker may have to study the service's vulnerabilities, perform some kind of scouting before really launching attacks or compromising the service. In addition, the residence time of an attacker in a state of the SMP for exploring or exploiting the vulnerabilities of service S can be cut off thanks to continuous cleansing cycle of SCIT. Thus, it is better that the modeling does not have to be constrained by an exponential function sojourn times. Within the QFITS, this analysis represents the first step of the framework that would allow the system designer to obtain expressions of steady probabilities necessary to compute certain IT-QoS parameters.

Now, let us establish the Semi-Markov model of an atomic service S protected by SCIT. Note that I do not consider a single service instance, but the entire service cluster hosted by SCIT nodes. Since SCIT does not contain any detection component, the state diagram can be modeled with three states as shown in Figure 4.1:

- G (Good): Service S starts with a Good state in which it functions normally.
- A (Attacked): As vulnerabilities are exploited, Service S is under attack.
- F (Fail): If the attack is successful, then Service S falls into Failure mode.

Thus, the state space contains three elements:

$$E = \{G, A, F\}$$

In terms of transition probabilities, the transition diagram shows:

- P_A as the probability that an attack occurs so that service S goes from state G to state A.
- P_C as the probability of that the online service instance being under attack goes to cleansing mode, and is replaced by a pristine service instance. This makes service S recover to state G from state A. While under attack in state A, service S can still operate; it might also be possible that an attack ceases without causing security failure to the Service.
-

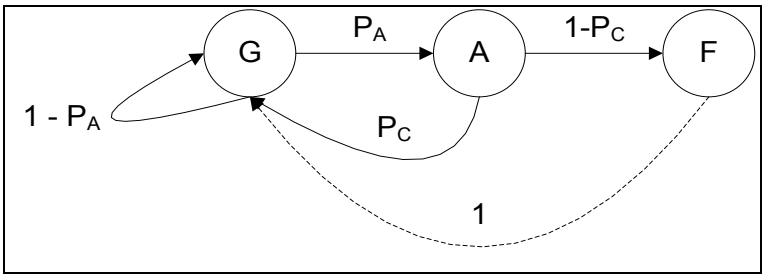

Figure 4.1. Transition Diagram for Simple SCIT

In [Ma2004, Ng2009], the transition diagram has state V between states G and A to capture the exploitation of vulnerabilities. In order to simplify the model, I choose to collapse states G and V to a single state G [Ng2012]. Some authors suggested to merge states G and A [Kre2010][Ng2010], which makes sense from the operational point of

view. But, in the model of this thesis, state A represents an intermediary step between state G and state F, and allows the introduction of the attack probability.

The transition probability matrix **P** can be written as:

$$\boldsymbol{P} = \begin{matrix} \\ G \\ A \\ F \end{matrix} \begin{matrix} \begin{matrix} G & A & F \end{matrix} \\ \begin{pmatrix} \overline{P_A} & P_A & 0 \\ P_C & 0 & \overline{P_C} \\ 1 & 0 & 0 \end{pmatrix} \end{matrix},$$

where $\overline{P_A} = 1 - P_A$ and $\overline{P_C} = 1 - P_C$.

4.4 MTTSF Analysis

As part of Module 2 in QFITS, I will solve the Semi-Markov Process in order to obtain the expression of MTTSF in terms of the probabilities P_A and P_C. Then, I will study how MTTSF varies with respect to these two probabilities.

4.4.1 MTTSF Expression

In order to compute the Mean Time to Security Failure (MTTSF), state F is considered as an absorbing one. But, in the SCIT architecture, state F has a low chance of occurring. However, there may be cases where state F is really absorbing:

- One possibility is when the Controller itself fails due to a hardware fault. Since the controller presents a single point of failure, the rotational cleansing algorithm is no longer executed for the nodes, causing a compromise to the service instance being in operation.
- It could also be that the attack on the active node hosting the service penetrates the virtual machine and/or the host machine so that the latter cannot respond to the signal of the Controller. Hence, the cleansing algorithm is interrupted.

Within this context, the state pace E is divided into the set of transient states E_T and the set of absorbing states E_A:

$$E_T = \{G, A\}, \text{ and}$$

$$E_A = \{F\}.$$

With these sets E_T and E_A, the transition probability matrix **Q** becomes:

$$\boldsymbol{Q} = \begin{matrix} \\ G \\ A \end{matrix} \begin{matrix} \begin{matrix} G & A \end{matrix} \\ \begin{pmatrix} \overline{P_A} & P_A \\ P_C & 0 \end{pmatrix} \end{matrix}.$$

Let:

- $\mathbf{X} = (X_G, X_A)$ be the row vector containing the number of visits at the transient states G and A in E_T;
- $\mathbf{H} = (H_G, H_A)$ the row vector containing the mean sojourn times at the transient states G and A in E_T;
- $\mathbf{X_0} = (1, 0)$ being the starting vector.

Then, the number of visits X can be found by solving the system of equations using the methodology described in [Lim2001, Ma2004]:

$$\mathbf{X} = \mathbf{X_0} + \mathbf{XQ} \qquad (4.1), \text{ or}$$

$$(X_G, X_A) = (1, 0) + (X_G, X_A)\begin{pmatrix} \overline{P_A} & P_A \\ P_C & 0 \end{pmatrix}$$

Solving (4.1) yields the expression for **X**:

$$\begin{cases} X_G = \dfrac{1}{P_A \overline{P_C}} \\ X_A = \dfrac{1}{\overline{P_C}} \end{cases} \qquad (4.2)$$

The expression for MTTSF is given by the scalar product:

$$\text{MTTSF} = \mathbf{X} . \mathbf{H}$$

Then, plugging the solution for **X** listed in (4.2) gives us the following expression for MTTSF:

$$\text{MTTSF} = \frac{H_G + P_A H_A}{P_A \overline{P_C}} \qquad (4.3)$$

4.4.2 Variation of MTTSF

Computing the partial derivatives of MTTSF in (4.3) with respect to P_A and P_C gives:

$$\frac{\partial(\text{MTTSF})}{\partial P_A} = -\frac{H_G}{P_A^2 \overline{P_C^2}} \text{ and}$$

$$\frac{\partial(\text{MTTSF})}{\partial P_C} = \frac{H_G + P_A H_A}{P_A \overline{P_C^2}}.$$

From these expressions, it is obvious that:

$$\frac{\partial(\text{MTTSF})}{\partial P_A} < 0 \text{ and } \frac{\partial(\text{MTTSF})}{\partial P_C} > 0.$$

Consequently, I can increase the MTTSF of service S by decreasing P_A somehow and increasing the cleansing probability P_C. So, the question is how to decrease P_A and increase P_C.

4.5 S-Availability Analysis

As part of Module 2 in QFITS, I will solve the Semi-Markov Process in order to obtain the expression of S-Availability (SA) in terms of the probabilities P_A and P_C. Then, I will study how SA varies with respect to these two probabilities.

4.5.1 S-Availability Expression

To derive the expression for the SA from security failures, I do not consider state F as an absorbing state [Ma2004]. Therefore, the entire transition matrix **P** encompassing all states in E = {G, A, F} will be used.

Let:

- **X** = (X_G X_A X_F) be the row vector containing the number of visits of states G, A and F in E;
- **H** = (H_G, H_A, H_F) the row vector containing the mean sojourn times at the states G, A and F in E.

The number of visits can be found by solving the system of equations:

$$\begin{cases} \mathbf{X} = \mathbf{XP} \\ \sum_{i \in E} X_i = 1 \end{cases} \qquad (4.4)$$

$$\Leftrightarrow \begin{cases} X_G = \overline{P_A} X_G + P_C X_A + X_F \\ X_A = \overline{P_A} X_G \\ X_F = \overline{P_C} X_A \\ X_G + X_A + X_F = 1 \end{cases}$$

Solving (4.4) yields the expression for **X**:

$$\begin{cases} X_G = \dfrac{1}{1 + P_A + P_A \overline{P_C}} \\ X_A = P_A X_G \\ X_F = P_A \overline{P_C} X_G \end{cases} \qquad (4.5)$$

Since SA is the time proportion spent in states other than state F, SA can be given by the formula [Ros2009, Tri2002, Ma2004]:

$$SA = 1 - \frac{X_F H_F}{\mathbf{X}.\mathbf{H}} \qquad (4.6).$$

Plugging the solution for **X** expressed by (4.5) into (4.6) yields the following expression for SA:

$$SA = 1 - \frac{P_A \bar{P_C} H_F}{H_G + P_A H_A + P_A \bar{P_C} H_F} \qquad (4.7).$$

$$SA = \frac{H_G + P_A H_A}{H_G + P_A H_A + P_A \bar{P_C} H_F} \qquad (4.7).$$

4.5.2 Variation of S-Availability

Computing the partial derivatives of SA in (4.7) with respect to P_A and P_C gives:

$$\frac{\partial(SA)}{\partial P_A} = -\frac{\bar{P_C} H_F H_G}{\Delta^2} \text{ and}$$

$$\frac{\partial(SA)}{\partial P_C} = \frac{P_A H_F (H_G + P_A H_A)}{\Delta^2},$$

where $\Delta^2 = H_G + P_A H_A + P_A \bar{P_C} H_F$.

From these expressions, it is obvious that:

$$\frac{\partial(SA)}{\partial P_A} < 0 \text{ and } \frac{\partial(SA)}{\partial P_C} > 0.$$

Similar to the variation analysis of MTTSF above, I can increase the SA value of service S by decreasing P_A somehow and increasing the cleansing probability P_C. I then arrive to the same question before: how to decrease P_A and increase P_C?

4.6 MTTSR Expression

In the case of an atomic service S protected by the simple SCIT scheme, the MTTSR is the mean time spent in state F, which is precisely the mean sojourn time in state F, denoted by H_F. Since the distribution function in state F is given by:

$$h_F(t) = \begin{cases} 1 - \frac{t}{W} & t \leq W \\ 1 & t > W \end{cases}.$$

It is obvious that the mean sojourn time H_F is:

$$H_F = \frac{W}{2}.$$

Thus, MTTSR depends solely on the exposure window:

$$\text{MTTSR} = \frac{W}{2} \quad (4.8).$$

4.7 S-Reliability Analysis

In this section, I will establish the expression of S-Reliability in terms the probabilities P_A and P_C. Then, the variation of S-Reliability with respect to these two probabilities will be studied.

4.7.1 S-Reliability Expression

Adapting the mathematical definition of reliability in [Lim2001, Pha2006], the S-Reliability of a service can be expressed by:

$$SR(t) = e^{-\int_0^t \lambda(u)du}$$

where $\lambda(u)$ is the failure rate function.

If I assume that the failure rate is constant with rate equal to MTTSF, then S-Reliability can be expressed in terms of MTTSF:

$$SR(t) = e^{-\frac{t}{MTTSF}}.$$

If I consider a period T (T can be a month or year for example), then I can write the S-Reliability for the period T as:

$$SR(T) = e^{-\frac{T}{MTTSF}} \quad (4.9).$$

4.7.2 Variation of S-Reliability

Computing the partial derivative of *Av* in with respect to P_A gives:

$$\frac{\partial(SR(T))}{\partial \mathrm{P_A}} = \left(e^{-\frac{T}{MTTSF}}\right)\left(\frac{T}{MTTSF^2}\right)\left(\frac{\partial(\mathrm{MTTSF})}{\partial \mathrm{P_A}}\right)$$

Since

$$\frac{\partial(\mathrm{MTTSF})}{\partial \mathrm{P_A}} < 0,$$

I obtain:

$$\frac{\partial(SR(T))}{\partial \mathrm{P_A}} < 0.$$

Similarly, the partial derivative of *SR(T)* with respect to P_C gives:

$$\frac{\partial(SR(T))}{\partial \mathrm{P_C}} = \left(e^{-\frac{T}{MTTSF}}\right)\left(\frac{T}{(MTTSF)^2}\right)\left(\frac{\partial(\mathrm{MTTSF})}{\partial \mathrm{P_C}}\right)$$

Since

$$\frac{\partial(\mathrm{MTTSF})}{\partial \mathrm{P_C}} > 0,$$

I obtain:

$$\frac{\partial(SR(T))}{\partial \mathrm{P_C}} < 0.$$

Thus, I can increase the S-Reliability if I can decrease the attack probability P_A and increase the cleansing probability P_C.

4.8 General Correlation Theorem

In what follows, I would like to establish the relationship between the exposure window W and the IT-QoS parameters previously analyzed. Such a relationship will provide a quantitative mechanism to control or adjust the above IT-QoS parameters via the exposure window. First, I will establish two Theorems treating the general case of a continuous and increasing cumulative distributive function (cdf). Then, I will consider special cases of the Poisson and Weibull models.

Since a SCIT node is online for only W_O time units, taking offline a SCIT node can be modeled by a uniform distribution with parameter W_O. A SCIT node vulnerability window is also a uniform distribution, but with parameter $W = W_O + W_{GP}$ due to the grace period.

4.8.1 Attack Probability

Let us compute the attack probability P_A, which is also the transition probability from state G to state A.

Theorem 4.1. Let S be a service protected by SCIT with active window W_O. Assume that the attack arrival has a continuous and strictly increasing cdf $F_X(t)$. Then the attack probability P_A is given by:

$$P_A = \frac{1}{W_O}\int_0^{W_O} F_X(t)dt.$$

Moreover, P_A decreases as W_O decreases.

Proof.

Before proceeding to the proof, let us note that F_X is a non-decreasing function by the nature of a cdf. Moreover, if F_X is not increasing, then it must be constant. The latter case is not interesting, since F_X will be bounded by default without any intervention of SCIT window. That is why the Theorem assumes that F_X is increasing.

Let X and Y be random variables for the times of the attack and cleansing events respectively. Let $F_X(t)$ and $F_Y(t)$ denote the respective cdf(s) of X and Y, with $F_X(t)$ being continuous. Since the cleansing rotation is deterministic and periodic, $F_Y(t)$ is given by the uniform distribution:

$$F_Y(t) = \begin{cases} \frac{t}{W_O}, & t < W_O \\ 1, & t \geq W_O \end{cases}.$$

The associated probability density function (pdf) $f_Y(t)$ is given by:

$$f_Y(t) = \frac{1}{W_O}.$$

When the attack event occurs before the cleansing one, the service protected by SCIT goes from state G to state A with probability $P_A = P(X < Y)$, i.e. when the attack event occurs before the cleansing one. Applying the method in [Tri2002], I can compute the probability P_A:

$$P_A = \int_0^{W_O} P(X < t) f_Y(t) dt$$

$$= \int_0^{W_O} P(X < t) \frac{1}{W_O} dt$$

$$= \frac{1}{W_O} \int_0^{W_O} F_X(t) dt.$$

According to the Mean Value Theorem for Integrals, I can find c such that:

$$F_X(c) = \frac{1}{W_O}\int_0^{W_O} F_X(t)dt, \quad c \in [0, W_O].$$

As W_O decreases, c decreases. Hence $F_X(c) = P_A$ decreases, since F_X is an increasing function.

QED.

4.8.2 Cleansing Probability

Let us compute the cleansing probability P_C when the service is in state A, so that the service can transition back to the good state G.

Theorem 4.2. Let S be a service protected by SCIT with exposure window W. the time that a malicious attack takes to compromise a service has a continuous and strictly increasing cdf denoted by $F_Z(t)$. Then the cleansing probability P_C is given by:

$$P_C = 1 - \frac{1}{W}\int_0^{W} F_Z(t)dt.$$

Moreover, P_C increases as W_O decreases.

Proof.

First, note that while the active window W_O is used in the correlation in Theorem 4.1, the exposure window, including the active window and the grace period window, should be used here for the correlation. The reason is that although the service in grace period does not accept additional transaction requests, it is still processes the requests that are already enqueued in the system. Let Z and Y be random variables for the times of the

compromise and cleansing events respectively. Let $F_Z(t)$ and $F_Y(t)$ denote the respective cdf(s) of Z and Y, with $F_Z(t)$ being continuous. In the Proof of Theorem 4.1, we have seen that $F_Y(t)$ is a uniform distribution within the interval [0,W].

When the compromise event occurs before the cleansing one, the service protected by SCIT goes from state A to state F with probability $\bar{P}_C = P(Z < Y)$.

Applying the method in [Tri2002], I can write:

$$\bar{P}_C = \int_0^W P(Z < t) f_Y(t) dt$$

$$= \int_0^W P(Z < t) \frac{1}{W} dt$$

$$= \frac{1}{W} \int_0^W F_Z(t) dt.$$

According the Mean Value Theorem for Integrals, I can find *c* such that:

$$F_Z(c) = \frac{1}{W} \int_0^W F_z(t) dt, \quad c \in [0, W].$$

As W decreases, c decreases. Hence, $F_Z(c)$ decreases, since it is an increasing function. Then: $P_C = 1 - \overline{P_C} = 1 - F_Z(c)$ increases, as W decreases.

QED.

4.9 Poisson Attack Arrival

In this section, I will establish the dependency of the probabilities P_A and P_C on the exposure window W, in the case the attack arrivals follow the Poisson process.

4.9.1 Correlation between W and {P_A, P_C}

Preliminary results based on Poisson model and exponential distribution for attack arrival have been published in [Ng2009]. With the assumption that the attacks are independent of each other instead of being mounted in stages, I can model them with a Poisson process. Thus, the sojourn time in the state G is an exponential distribution with parameter λ_A, which is the attack arrival rate. For simplicity, the sojourn time in state A is modeled by the exponential distribution with parameter λ_F. From state F, the service will go back to the G state due to periodic cleansing. At any state, the uniform distribution of the periodic cleansing will influence the sojourn time in that state, including failure state F.

Theorem 4.3. Assuming that the attack arrival is a Poisson process with rate λ_A, and the SCIT exposure window is W_O, the attack probability P_A is given by:

$$P_A = 1 - \frac{1}{\lambda_A W_O}\left(1 - e^{-\lambda_A W_O}\right) \quad (4.10).$$

Proof.

Let X and Y be the random variables for the times of the attack and cleansing events respectively. The respective cdf(s) of X and Y are exponential and uniform distributions. The service protected by SCIT goes from state G to state A with probability P_A such that: $P_A = P(X < Y)$. Applying the method in [Tri2002]:

$$P_A = \int_0^{W_O} P(X < t) f_Y(t) dt$$

$$= \int_0^{W_O} (1 - e^{-\lambda_A t}) \frac{1}{W_O} dt$$

$$= \frac{1}{W_O}\left[t + \frac{1}{\lambda_A} e^{-\lambda_A t}\right]_0^{W_O}$$

$$= 1 - \frac{1}{\lambda_A W_O}\left(1 - e^{-\lambda_A W_O}\right).$$

QED.

Using the same random variables X and Y of Theorem 4.1, the cdf of X is:

$F_X(t) = 1 - e^{-\lambda_A t}$.

As $F_X(t)$ is clearly increasing, Theorem 4.1 helps us to state that the attack probability P_A decreases as the active window decreases. But, one can also prove this correlation by taking the partial derivative of P_A in the special case of Poisson model with respect to W_O. Indeed:

$$\frac{\partial P_A}{\partial W_O} = -\frac{1}{\lambda_A {W_O}^2}\left[\lambda_A W_O e^{-\lambda_A W_O} - \left(1 - e^{-\lambda_A W_O}\right)\right]$$

$$\frac{\partial P_A}{\partial W_O} = -\frac{1}{\lambda_A {W_O}^2 e^{\lambda_A W_O}}\left[\lambda_A W_O - e^{\lambda_A W_O} + 1\right]$$

$$\frac{\partial P_A}{\partial W_O} = \frac{1}{\lambda_A {W_O}^2 e^{-\lambda_A W_O}}\left[e^{\lambda_A W_O} - (\lambda_A W_O + 1)\right].$$

The mean-value theorem applied for the function e^x gives:

$$\frac{e^{\lambda_A W_O} - 1}{\lambda_A W_O} > 1 \text{ for } \lambda_A W_O > 0.$$

Hence:

$$e^{\lambda_A W_O} - (\lambda_A W_O + 1) > 0,$$

i.e.:

$$\frac{\partial \mathrm{P_A}}{\partial \mathrm{W_O}} > 0.$$

Hence, the correlation between $\mathrm{P_A}$ and $\mathrm{W_O}$ is proved, i.e. $\mathrm{P_A}$ decreases as $\mathrm{W_O}$ decreases.

Theorem 4.4. Assuming that the time that a malicious attack takes to compromise the service is modeled as an exponential distribution with rate λ_F, and the SCIT exposure window is W, the attack probability $\mathrm{P_C}$ is given by:

$$P_C = \frac{1}{\lambda_F W}\left(1 - e^{-\lambda_F W}\right) \quad (4.11).$$

Proof.

First, note that the window of compromise is not just the active window $\mathrm{W_O}$, but the exposure window including the grace period. Let Z and Y be the random variables for the times of the compromise and cleansing events respectively. The respective cdf(s) of Z and Y are exponential and uniform distributions. If I compute $\bar{P}_C$, and derive $\mathrm{P_C}$, then the proof will be exactly the one above. The service will be compromised and goes from state A to state F with probability $\bar{P}_C = P(Z < Y)$, i.e. when the cleansing happens before the attacker can compromise the service. Applying the methods in [Tri2002]:

$$\overline{P_C} = \int_0^W P(Z < t) f_Y(t) dt$$

$$= \int_0^W (1 - e^{-\lambda_F t}) \frac{1}{W} dt$$

$$= \frac{1}{W}\left[t + \frac{1}{\lambda_F} e^{-\lambda_F t}\right]_0^W$$

$$= 1 - \frac{1}{\lambda_F W}\left(1 - e^{-\lambda_F W}\right).$$

Then:

$$P_C = \frac{1}{\lambda_F W}\left(1 - e^{-\lambda_F W}\right).$$

QED.

Note that the cleansing probability P_C increases as the exposure window decreases, according to Theorem 4.2. One can also prove this correlation directly by using differentiation. The partial derivative of P_C with respect to W gives:

$$\frac{\partial \mathrm{P_C}}{\partial \mathrm{W}} = \frac{1}{\lambda_F W^2}\left[\lambda_F W e^{-\lambda_F W} - \left(1 - e^{-\lambda_F W}\right)\right]$$

$$\frac{\partial \mathrm{P_C}}{\partial \mathrm{W}} = -\frac{1}{\lambda_F W^2 e^{\lambda_F W}}\left[e^{\lambda_F W} - 1 - \lambda_F W\right]$$

It can be proved by using the mean-value theorem that:

$$\frac{e^{\lambda_F W} - 1}{\lambda_F W} > 1 \text{ for } \lambda_F W > 0, \text{i. e}$$

$$e^{\lambda_F W} - 1 - \lambda_F W > 0.$$

Hence:

$$\frac{\partial \mathrm{P_C}}{\partial \mathrm{W}} < 0.$$

Then, the cleansing probability that can get the service back to the G(ood) state will increase if the window decreases.

QED.

Assume that the attack arrival is a Poisson process with rate λ_A, and the compromise time follows the exponential distribution with rate λ_F. Applying the methods in [Tri2002], the distribution functions of the sojourn times at G, A and F are given by:

$$\begin{cases} h_G(t) = \begin{cases} 1 - \left(1 - \frac{t}{W_O}\right) e^{-\lambda_A t} & t < W_O \\ 1 & t \geq W_O \end{cases} \\ h_A(t) = \begin{cases} 1 - \left(1 - \frac{t}{W}\right) e^{-\lambda_F t} & t < W \\ 1 & t \geq W \end{cases} \\ h_F(t) = \begin{cases} 1 - \frac{t}{W} & t < W \\ 1 & t \geq W \end{cases} \end{cases} \qquad (4.12)$$

Proposition 4.5. If the sojourn times at states G, A and F are given by (4.12), then their respective mean sojourn times at the states G, A, and F are given by:

$$\begin{cases} H_G = \frac{1}{\lambda_A} - \frac{1}{{\lambda_A}^2 W_O}(1 - e^{-\lambda_A W_O}) \\ H_A = \frac{1}{\lambda_F} - \frac{1}{{\lambda_F}^2 W}(1 - e^{-\lambda_F W}) \\ H_F = \frac{W}{2} \end{cases} \qquad (4.13).$$

Proof.

The first two expressions in Proposition 4.5 are simply applications of the methods and results in [Tri2002].

$$\mathrm{H_G} = \int_0^{\infty} (1 - \mathrm{h_G(t)})\mathrm{dt}$$

$$\int_0^{\mathrm{W_O}} \left(1 - \frac{t}{W_O}\right) e^{-\lambda_A t} dt$$

$$= \left[-\frac{1}{\lambda_A} e^{-\lambda_A t} + \frac{1}{{\lambda_A}^2 \mathrm{W_O}} (1 + \lambda_A t) e^{-\lambda_A t} \right]_0^{\mathrm{W_O}}$$

$$= -\frac{1}{\lambda_A} e^{-\lambda_A W_O} + \frac{1}{{\lambda_A}^2 W_O}(1 + \lambda_A W_O) e^{-\lambda_A W_O} + \frac{1}{\lambda_A} - \frac{1}{{\lambda_A}^2 W_O}$$

$$= \frac{1}{\lambda_A} - \frac{1}{{\lambda_A}^2 W_O}\left(1 - e^{-\lambda_A W_O}\right).$$

Similar calculations will yield the expression for H_A as stated in Proposition 4.5.

$$H_F = \int_0^\infty (1 - h_F(t))dt$$

$$H_F = \int_0^\infty \frac{t}{W} dt$$

$$= \left[\frac{t^2}{2W}\right]_0^W = \frac{W}{2}.$$

4.9.2 Summary

To summarize, I can state the following:

- When the window decreases, the attack probability P_A decreases and the cleansing probability P_C increases.
- From the variation analysis in previous sections, I can imply that if W or W_O decreases:
    - MTTSF will increase
    - MTTSR will decrease
    - S-Availability Av will increase
    - S-Reliability SR will increase.

By combining (4.3), (4.10) and (4.11), I can express MTTSF as an algebraic function with variables W and W_O:

MTTSF

$$= \frac{\frac{1}{\lambda_A} - \frac{1}{{\lambda_A}^2 W_O}(1 - e^{-\lambda_A W_O}) + \left(1 - \frac{1}{\lambda_A W_O}(1 - e^{-\lambda_A W_O})\right)\left(1 - \frac{1}{\lambda_F W}(1 - e^{-\lambda_F W})\right)}{\left(1 - \frac{1}{\lambda_A W_O}(1 - e^{-\lambda_A W_O})\right)\left(1 - \frac{1}{\lambda_F W}(1 - e^{-\lambda_F W})\right)} \quad (4.14).$$

Similarly, I can express the other IT-QoS, namely MTTSR, SA, and SR in terms of the SCIT parameters W, and W_O, the attack arrival rate λ_A, and the compromise rate λ_F.

This result is significant in the sense that it makes the SCIT exposure window W_O a tuning parameter for achieving the desired level(s) of IT-QoS parameters for a service.

4.9.3 Numerical Examples

To illustrate the effects of the exposure window on the probabilities P_A and P_C, and hence the IT-QoS that I have previously analyzed, let us consider some numerical examples.

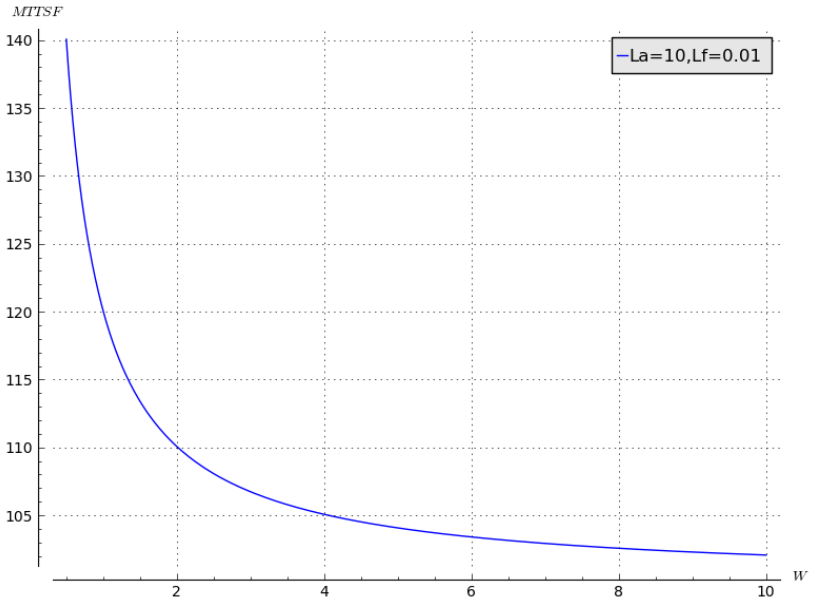

Figure 4.2. Simple SCIT - λ_A=10, λ_F =0.01.

Figure 4.2 visualizes the variation pattern of MTTSF with respect to W. If we let a minute as a unit of time, then the attack arrival rate λ_A means 10 attacks per minute or about 14400 a day. The compromise rate λ_F = 0.01 is equivalent to the time it takes to compromise a service is about 1 hour.

The following Figure shows the relative change in behavior for different values of attack arrival rate and compromise rate. Decimal logarithm of MTTSF is used so that the curves corresponding to different orders of magnitude can be displayed in the same graph. In the figure, La and Lf denote λ_A and λ_A respectively.

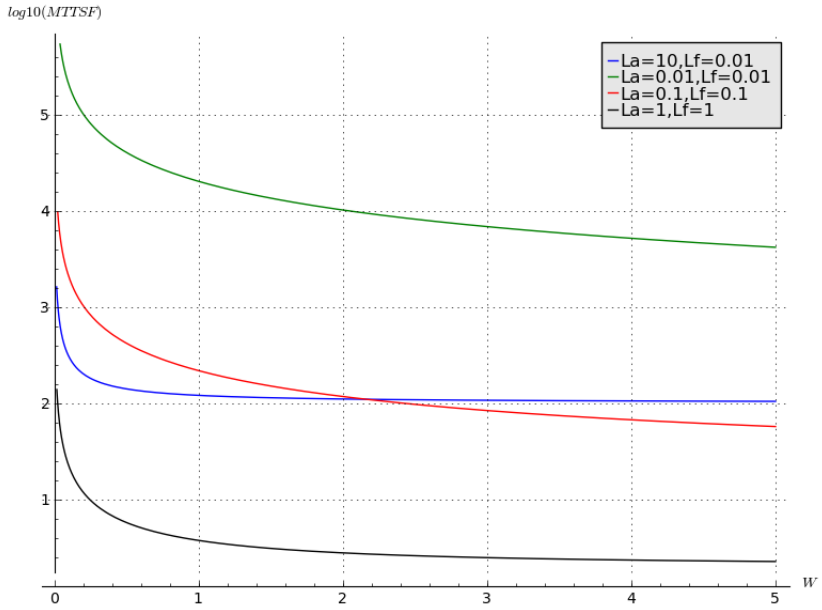

Figure 4.3. Simple SCIT - Various values for λ_A and λ_F.

Note that in the case where $\lambda_A = \lambda_F = 1$, the exposure W = 1 only yields a MTTSF about 5 time units. This implies that if both attack rates and compromise rates are of the same order of magnitude than the exposure window, then the benefits are not significant.

4.9.4 Simulation

The Semi-Markov Process for a service protected by simple SCIT has been simulated using Java and the distribution functions in the Apache Commons Math 3.3 library. The Java code for the simulation is listed in Appendix A. The function **runUntilFail** is the main function in the simulation, and its pseudo-code is listed in Table 4.1. The inputs to the function are: the arrival rate λ_A, compromise rate λ_F, active window W_O, and exposure window W.

Table 4.1. Pseudo-code of simple SCIT SMP simulation.

```
runUntilFail (λA, λF)

1. state = G; ttsf = 0.0
2. while (state is not F)
3.       if (state is G)
4.            get attack inter-arrival time at from exponential distribution Exp(λA)
5.            get cleansing time clt from the uniform distribution U(WO)
6.            if (at < clt) then
7.                 go to state A
8.                update ttsf: ttsf = ttsf + at
9.            else
10.                update ttsf: ttsf = ttsf + clt
10.           endif
11.      else if (state is A)
12.           get time ct from compromise exponential distribution Exp(λF)
13.           get cleansing time clt from the uniform distribution U(W)
14.           if (ct < clt) then
15.                go to state F
16.                update ttsf: ttsf = ttsf + ct
17.           else
18.                go back to state G
```

```
19.              update ttsf: ttsf = ttsf + clt
20.          endif
21.       endif
22.       if (state is F)
23.          get sojourn time ttsr from the uniform distribution for cleansing
21. endwhile
22. return ttsf and ttsr
```

The function **runUntilFail** simulates the Semi-Markov process that a service S protected by simple SCIT goes through from state G to state F. The function starts with state G as in line 1. Then, it will iterate until state F is reached (line 2). State G changes to state A if the inter-arrival time *at* is less than the cleansing time *clt* (lines 6-7). While in state A (line 11), the service will move to state F if the compromise time *at* is less than the cleansing time *clt* (lines 14-15); otherwise, the service goes back to state G (line 18). The values for *ttsf* will be updated accordingly (lines 8, 10, 16, 19). The value for *ttsr* is computed in line 23, when state F has been reached.

At the end of the execution, the function **runUntilFail** returns *ttsf* and *ttsr*. The values *ttsf* and *ttsr* are respectively the time the process takes to go from state G to state F for one run, and the time to recover back from state F to state G. In the simulation, the function is executed with multiple iterations. The average of all the obtained values for *ttsf* is the value for $MTTSF_{sim}$ in the simulation. The values collected for *ttsr* are used to compute the simulated value for $MTTSR_{sim}$. The simulation value for SA_{sim} is then derived by using the formula:

$$SA_{sim} = \frac{MTTSF_{sim}}{MTTSF_{sim} + MTTSR_{sim}}$$

Thus, each value of $MTTSF_{sim}$ and SA_{sim} corresponds to a sample of data obtained via N = 10,000 iterations of the function **runUntilFail**. For the simulation, 10 samples were collected to plot the graph as shown in Figures 4.4 to 4.11, where the solid lines plot the analytical expressions for MTTSF and SA, and the scatter points represent the simulated data.

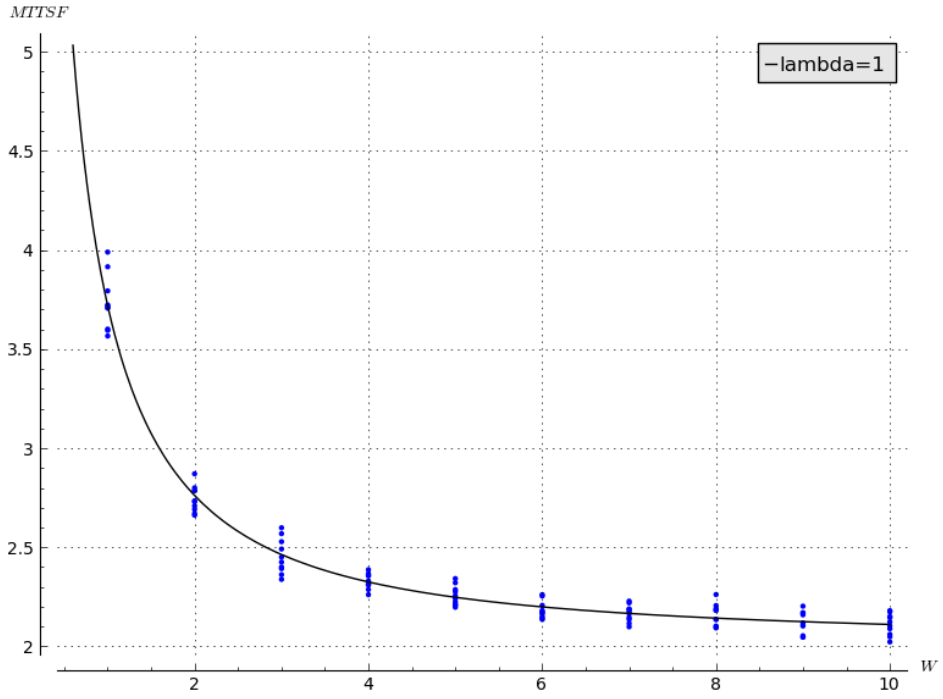

Figure 4.4. MTTSF of simple SCIT - λA, λF = 1.

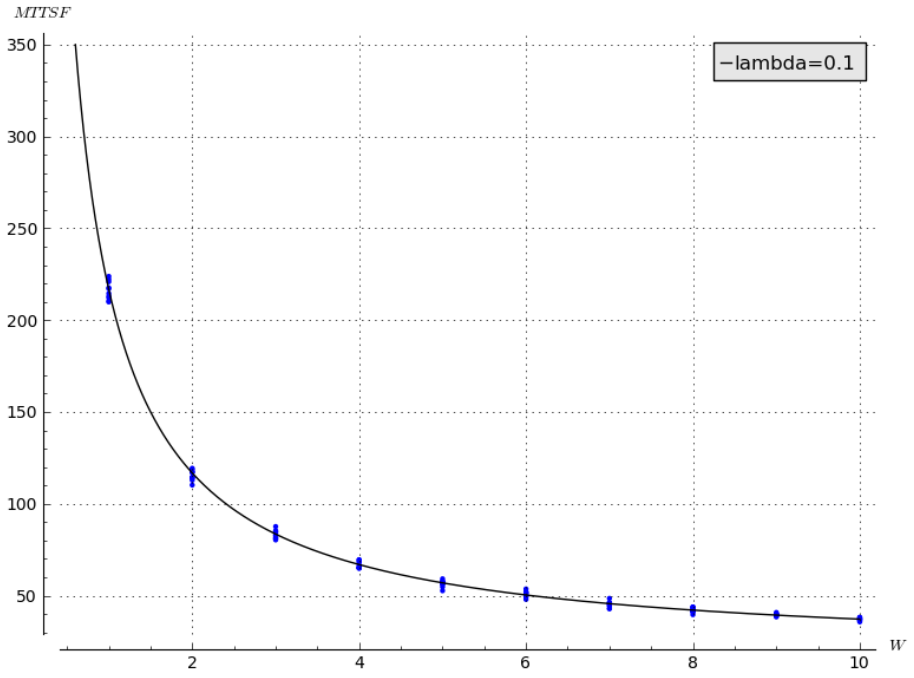

Figure 4.5. MTTSF of simple SCIT - λA, λF = 0.1.

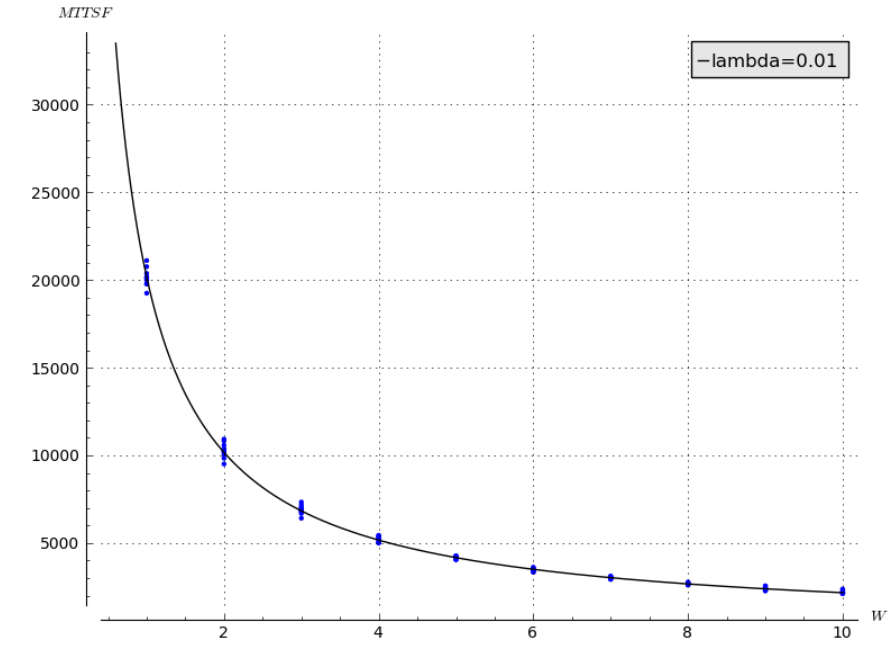

Figure 4.6. MTTSF of simple SCIT - λA, λF = 0.01.

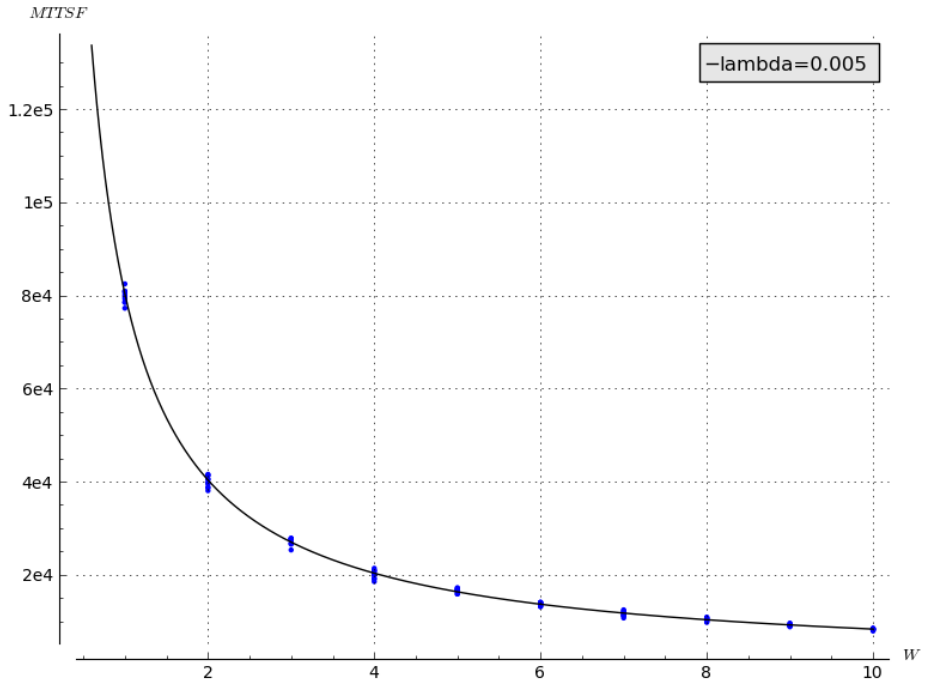

Figure 4.7. MTTSF of simple SCIT - λ_A, λ_F = 0.005.

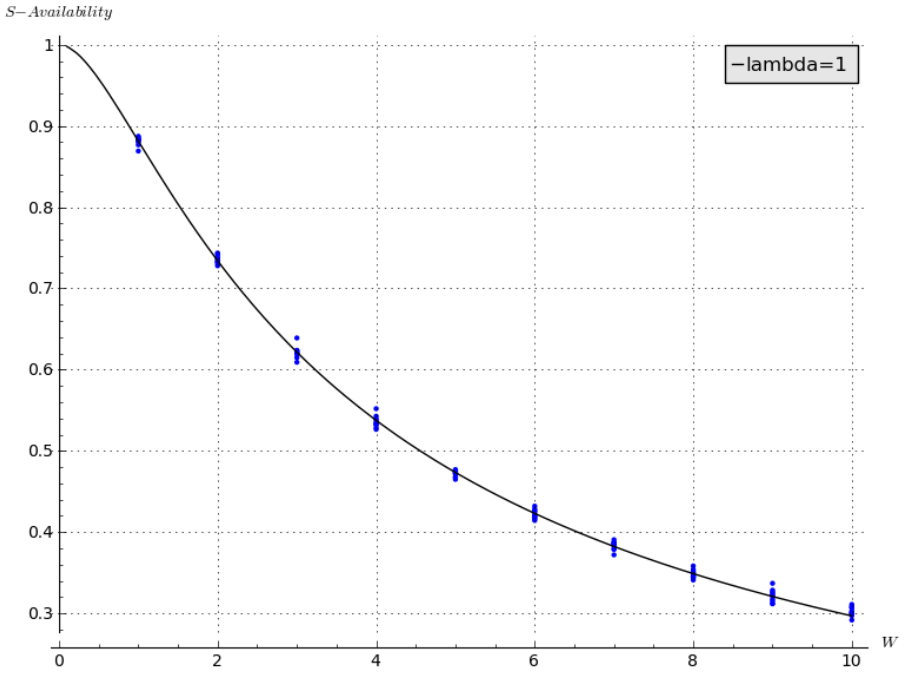

Figure 4.8. S-Availability of simple SCIT - λA, λF = 1.

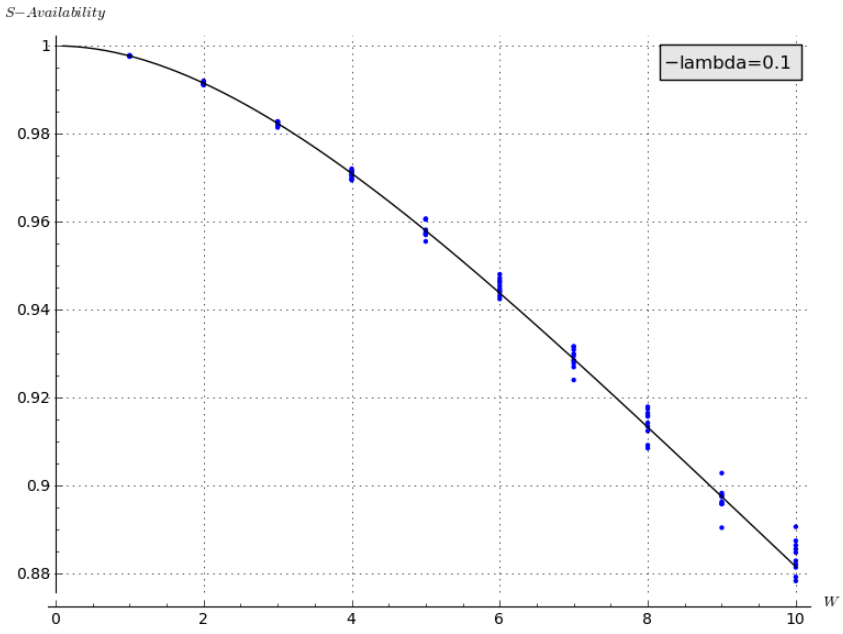

Figure 4.9. S-Availability of simple SCIT - λA, λF = 0.1.

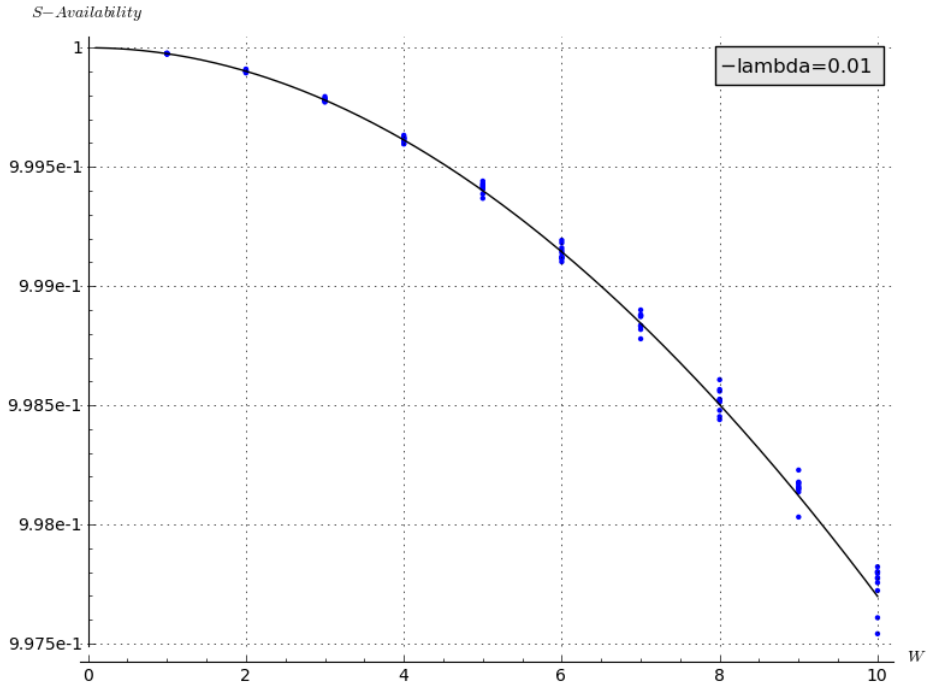

Figure 4.10. S-Availability of simple SCIT - λ_A, λ_F = 0.01.

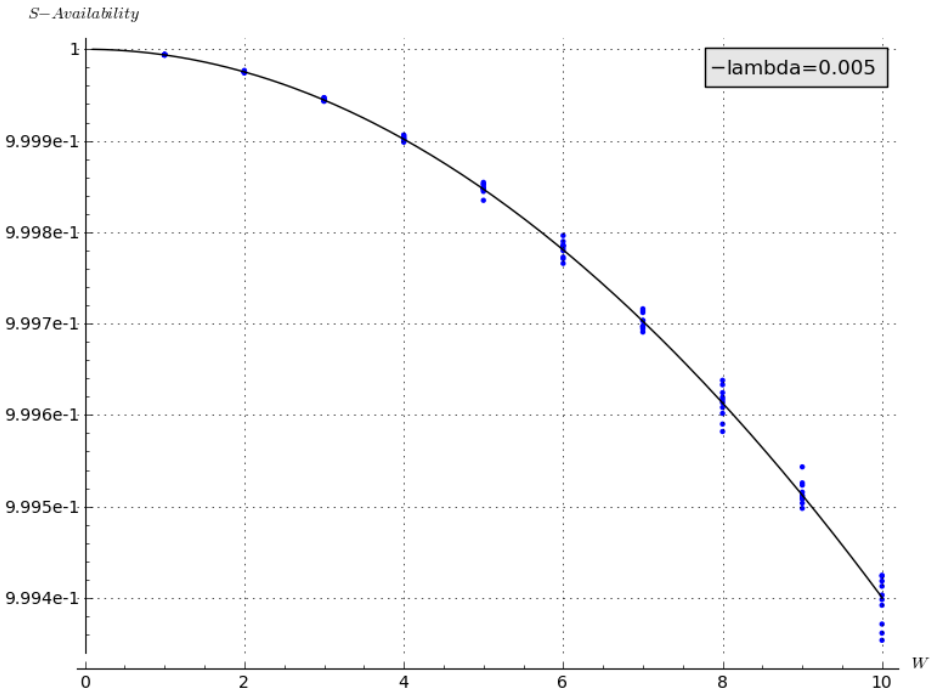

Figure 4.11. S-Availability of simple SCIT - λA, λF = 0.005.

4.10 Weibull Distribution for Attack Arrival

Now, I will study the probabilities of attack and cleansing if the attack arrival follows a Weibull distribution. From the mathematical point of view, the Weibull distribution will generalize the study, since the exponential distribution is a special case of Weibull in which the shape parameter is equal to 1. The shape parameter greater than one can be used to model the increasing rate of attack arrival; in this case, the attacks are mounted in phases such that the first ones are for scouting, and the later ones exploit the knowledge acquired during the scouting phase. The Weibull distribution has been used in the computation of the sojourn times in [Ma2004]. Note that for computing the analytical integral of an expression involving Weibull distribution, I have used Wolfram Mathematica Online Integrator [Wol2014].

4.10.1 Correlation between W and $\{P_A, P_C\}$

Attack Probability. With the Poisson model for the attack arrival, I was able to establish the relationship between the attack probability P_A and the SCIT exposure window W_O. Can I obtain similar results with the Weibull distribution with parameters λ_A and α which are the average number of attacks per unit time and the change rate respectively? With Weibull distribution, the inter-arrival time Y between two successive attacks will follow the probability:

$$F_Y(t) = P(Y \leq t) = 1 - e^{-\lambda_A t^{\alpha}}.$$

Theorem 4.6. If the inter-arrival time of attacks follows the Weibull distribution with parameters λ_A and γ_A, then the attack probability is given by:

$$P_A = 1 - (W_O \lambda_A^{\frac{1}{\alpha}} \alpha)^{-1}[\Gamma(\alpha^{-1}, 0) - \Gamma(\alpha^{-1}, \lambda_A {W_O}^{\alpha})] \qquad (4.15).$$

Proof.

Using the similar proof as for Theorem 4.3, I can write:

$$P_A = \int_0^{W_O} P(Y < t) f_X(t) dt$$

$$= \int_0^{W_O} (1 - e^{-\lambda_A t^{\alpha}}) \frac{1}{W_O} dt$$

$$= \frac{1}{W_O} \left[t + t(\lambda_A t^{\alpha})^{\frac{-1}{\alpha}} \alpha^{-1} \Gamma(\alpha^{-1}, \lambda_A t^{\alpha}) \right]_0^{W_O}$$

$$= \frac{1}{W_O} \left[t + {\lambda_A}^{-\frac{1}{\alpha}} \alpha^{-1} \Gamma(\alpha^{-1}, \lambda_A t^{\alpha}) \right]_0^{W_O}$$

$$= \frac{1}{W_O} \left[W_O + {\lambda_A}^{-\frac{1}{\alpha}} \alpha^{-1} \Gamma(\alpha^{-1}, \lambda_A {W_O}^{\alpha}) - {\lambda_A}^{-\frac{1}{\alpha}} \alpha^{-1} \Gamma(\alpha^{-1}, 0) \right]$$

$$= 1 - (W_O \lambda_A^{\frac{1}{\alpha}} \alpha)^{-1}[\Gamma(\alpha^{-1}, 0) - \Gamma(\alpha^{-1}, \lambda_A {W_O}^{\alpha})]$$

QED.

Cleansing Probability. Now I will study the relationship between the attack probability P_C and the SCIT exposure window W_O using the Weibull distribution for the compromise time in state A. Thus, the so-called compromise time Z is modeled by Weibull distribution with rate λ_F and shape rate β, then the cdf (cumulative distribution function) of Z can be written as:

$$F_Z(t) = P(Z \leq t) = 1 - e^{-\lambda_F t^{\beta}}.$$

Theorem 4.7. If the compromise time of attacks follows the Weibull distribution with parameters λ_F and γ, then the cleansing probability P_C is given by:

$$P_C = (W\lambda_F^{\frac{1}{\beta}}\beta)^{-1}\left[\Gamma(\beta^{-1},0) - \Gamma\left(\beta^{-1},\lambda_F W^{\beta}\right)\right] \qquad (4.16).$$

Proof.

Using the similar proof as for Theorem 4.4, I can write:

$$\bar{P}_C = \int_0^W P(Y<t) f_X(t)dt$$

$$= \int_0^W (1-e^{-\lambda_F t^{\beta}})\frac{1}{W}dt$$

$$= \frac{1}{W}\left[t + t(\lambda_F t^{\gamma})^{\frac{-1}{\beta}}\beta^{-1}\Gamma(\beta^{-1},\lambda_F t^{\beta})\right]_0^W$$

$$= \frac{1}{W}\left[t + \lambda_F{}^{-\frac{1}{\beta}}\beta^{-1}\Gamma(\beta^{-1},\lambda_F t^{\beta})\right]_0^W$$

$$= \frac{1}{W}\left[W + \lambda_F{}^{-\frac{1}{\beta}}\beta^{-1}\Gamma\left(\beta^{-1},\lambda_F W^{\beta}\right) - \lambda_F{}^{-\frac{1}{\beta}}\beta^{-1}\Gamma(\beta^{-1},0)\right]$$

$$= 1 - (W\lambda_F^{\frac{1}{\beta}}\beta)^{-1}\left[\Gamma(\beta^{-1},0) - \Gamma\left(\beta^{-1},\lambda_F W^{\beta}\right)\right]$$

Then:

$$P_C = (W\lambda_F^{\frac{1}{\beta}}\beta)^{-1}\left[\Gamma(\beta^{-1},0) - \Gamma\left(\beta^{-1},\lambda_F W^{\beta}\right)\right].$$

QED.

Assuming that the attack arrival follows a Weibull distribution with rate λ_A and shape α, and the compromise time the Weibull distribution with rate λ_F and shape β, then according to [Tri2002], the distribution functions of the sojourn times at G, A and F are given by:

$$\begin{cases} h_G(t) = \begin{cases} 1 - \left(1 - \dfrac{t}{W_O}\right) e^{-\lambda_A t^{\alpha}} & t < W_O \\ 1 & t \geq W_O \end{cases} \\ h_A(t) = \begin{cases} 1 - \left(1 - \dfrac{t}{W}\right) e^{-\lambda_F t^{\beta}} & t < W \\ 1 & t \geq W \end{cases} \\ h_F(t) = \begin{cases} 1 - \dfrac{t}{W} & t < W \\ 1 & t \geq W \end{cases} \end{cases} \tag{4.17}$$

Proposition 4.6. If the sojourn times at states G, A and F are given by (4.17), then their respective mean sojourn times at the states G, A, and F are given by:

$$\begin{cases} H_G = (\alpha \lambda_A^{\frac{1}{\alpha}})^{-1}[\Gamma(\alpha^{-1}, 0) - \Gamma(\alpha^{-1}, \lambda_A {W_O}^{\alpha})] - (\alpha W_O \lambda_A^{\frac{2}{\alpha}})^{-1}[\Gamma(2\alpha^{-1}, 0) - \Gamma(2\alpha^{-1}, \lambda_A {W_O}^{\alpha})] \\ H_A = (\beta \lambda_F^{\frac{1}{\beta}})^{-1}\left[\Gamma(\beta^{-1}, 0) - \Gamma\left(\beta^{-1}, \lambda_F {W_O}^{\beta}\right)\right] - (\beta W_O \lambda_F^{\frac{2}{\beta}})^{-1}[\Gamma(2\beta^{-1}, 0) - \Gamma\left(2\beta^{-1}, \lambda_F {W_O}^{\beta}\right)] \\ H_F = \dfrac{W}{2} \end{cases}$$
(4.18)

Proof.

The first two expressions in Proposition 4.6 are applications from the methods and results in [Tri2002].

$$H_G = \int_0^{\infty} (1 - h_G(t))dt$$

$$\int_0^{W_O} \left(1 - \frac{t}{W_O}\right) e^{-\lambda_A t^{\alpha}} dt$$

$$= (\alpha W_O)^{-1}\left[\lambda_A{}^{-\frac{2}{\alpha}}\left(\Gamma(2\alpha^{-1}, \lambda_A t^{\alpha}) - W_O\lambda_A{}^{\frac{-1}{\alpha}}\Gamma(\alpha^{-1}, \lambda_A t^{\alpha})\right]_0^{W_O}\right.$$

$$= (\alpha W_O)^{-1}[\lambda_A{}^{-\frac{2}{\alpha}}\left(\Gamma(2\alpha^{-1}, \lambda_A {W_O}^{\alpha}) - W_O\lambda_A{}^{-\frac{1}{\alpha}}\Gamma(\alpha^{-1}, \lambda_A {W_O}^{\alpha})\right.$$

$$\left. - \lambda_A{}^{-\frac{2}{\alpha}}\left(\Gamma(2\alpha^{-1}, 0) + W_O\lambda_A{}^{-\frac{1}{\alpha}}\Gamma(\alpha^{-1}, 0)\right]\right.$$

$$= (\alpha\lambda_A{}^{\frac{1}{\alpha}})^{-1}[\Gamma(\alpha^{-1}, 0) - \Gamma(\alpha^{-1}, \lambda_A {W_O}^{\alpha})] - (\alpha W_O\lambda_A{}^{\frac{2}{\alpha}})^{-1}[\Gamma(2\alpha^{-1}, 0)$$

$$- \Gamma(2\alpha^{-1}, \lambda_A {W_O}^{\alpha})]$$

Similar calculations will yield the expression for H_A, and H_F was already computed in the proof of Proposition 4.5.

4.10.2 Summary

From the variation analysis in previous sections, I can imply that if W_O decreases:

- MTTSF will increase
- MTTSR will decrease
- S-Availability SA will increase
- S-Reliability SR will increase.

By combining (4.3), (4.15) and (4.16), I can express the IT-QoS above as an algebraic function with variables W and W_O, as in the case of a Poisson model.

This result is significant in the sense that it makes the SCIT exposure window W_O a tuning parameter for achieving the desired level(s) of IT-QoS parameters for a service.

4.10.3 Numerical Examples

In the following examples, both the attack and compromise probabilities follow the same Weibull distribution of rate $\lambda_A = \lambda_F = 10^{-3}$ and shape $\alpha = \beta = 2$.

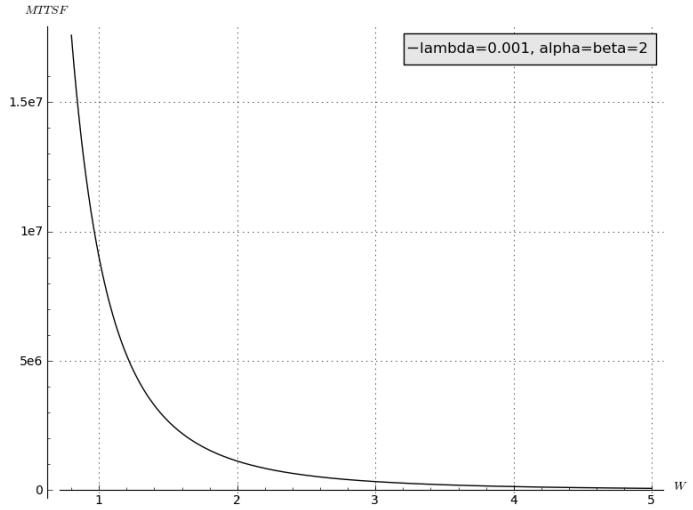

Figure 4.12. Simple SCIT - Weibull with shape = 2.

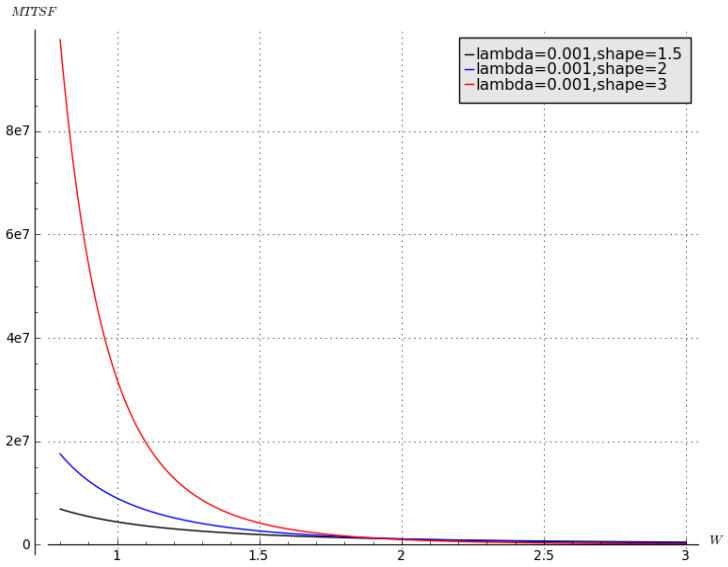

Figure 4.13. Simple SCIT - Weibull with various shape parameters.

From the two figures above, we can see that: a) the variation pattern of MTTSF with respect to W is as expected. Moreover, the shape increase diminishes the values of MTTSF for the same value of the exposure window.

CHAPTER 5 - SIMPLE SCIT AND COMPOSITE SERVICE

5.1 Overview

In the previous chapter, the analysis of an atomic service protected by SCIT has led to interesting results that allow the system designer to tune the exposure window in order to achieve the desired levels of the QoS metrics of interest. In this chapter, I will continue the same kind of analysis using Semi-Markov Process as applied to the case of a composite service. The computation of the steady-state probabilities the Semi-Markov chain as applied to the composite service protected by SCIT should allow me to derive:

- The expressions of a set of IT-QoS: MTTSF, MTTSR, S-Availability, and S-Reliability.
- The relationships between these IT-QoS and the SCIT control parameters, namely the exposure window, the grace period and number of SCIT nodes.

Composite services protected within the SCIT architecture was first studied in [Ng2010a], while the analysis of the special case with Enabler and Target services was described in [Ng2012].

5.2 Enabler Service Concept

The notion of "Enabler Service" was described in the study of attack surface in [How2002], as a service that has to be exploited and compromised by malicious actor before the latter can get to the target service. In the picture of defense-in-depth, the enabler service can be viewed as a line of defense for the target service. For instance, let us consider a search application composed of two services:

- User Input Service validating all user input criteria in the search requests,
- Search Service performing the actual search.

In this search application, every transition request to the Search Service must go through the User Input Service, which plays the role of the Enabler. For example, a SQL injection attack must compromise the User Input Service to launch attack on the Search service with its index repository.

Another example is the Authentication Service preceding the search application. In the case where Single Sign-On (SSO) is used for other applications as well, the user will be able to access these applications under the SSO domain once he/she authenticates successfully.

Thus, in an SOA environment, there are three types of scenarios as shown in the following figure: a) a pair of one enabler service and one target service; b) one enabler service and multiple target services type; c) multiple enabler services and a target service. The enabler services in scenario c) can be chained together in sequence or in parallel in front of the target service. If the enabler services are orchestrated in sequence, then one enabler will be the enabler of the following enabler; in this case, the analysis will be

similar to the scenario a). The key characteristic of the composite service with Enabler is that an attacker must compromise the enabler service S_1 before starting to attack the target service S.

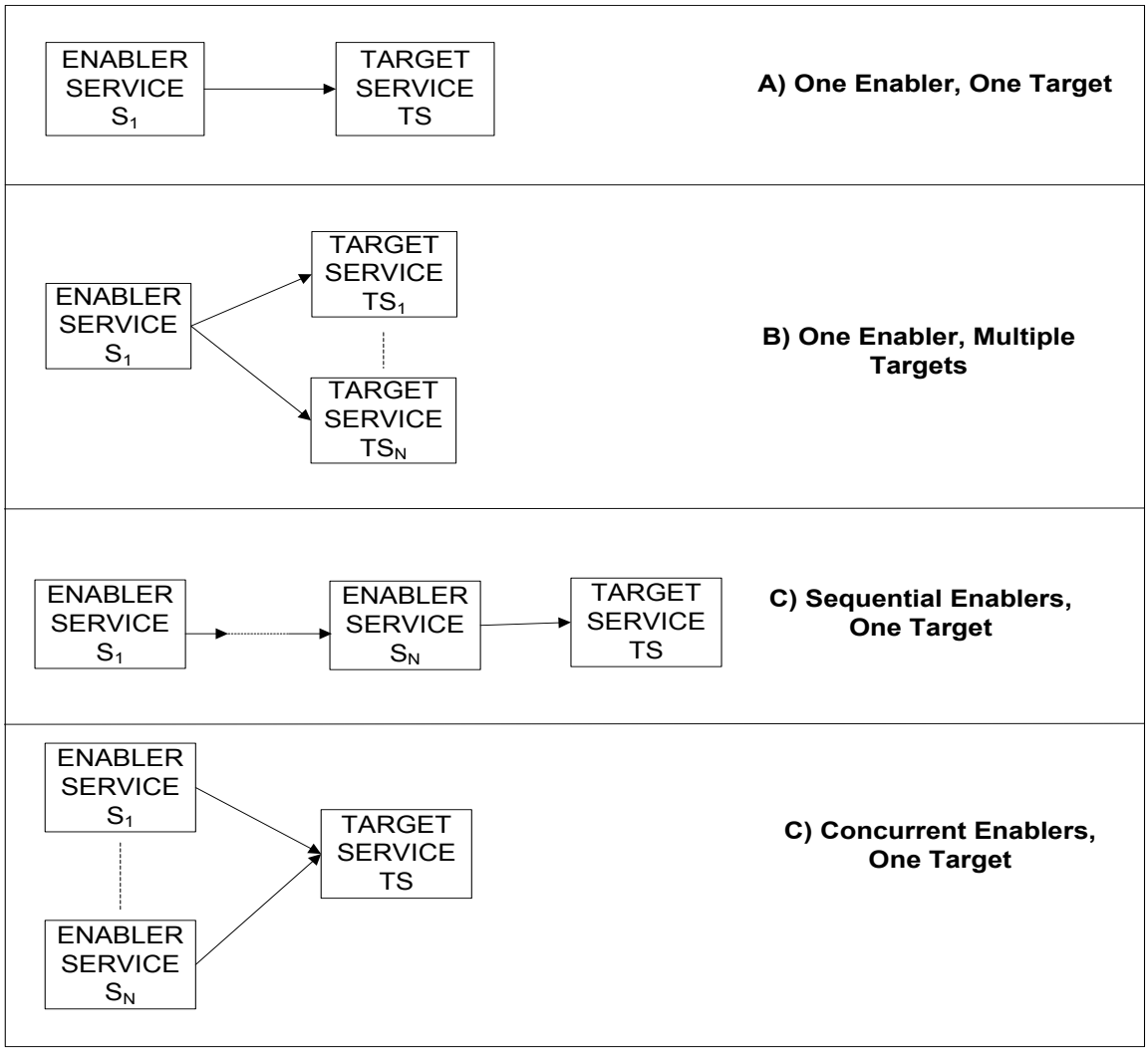

Figure 5.1. Scenario Types of Enabler Services

The analysis in this section will start by considering the first case of a pair of target service S protected by the enabler service S_1. For the case where there is one enabler S_1 and multiple targets TS_1, TS_2, ..., TS_N, the same reasoning can be applied for each pair (S_1, TS_i), with i = 1, 2, ...n. The remaining case of a target service protected by multiple enablers is a little more complicated; I conjecture that the resilience of the target service will depend on the most stringent enabler. The analysis that follows will help finding out what constitutes a "stringent enabler".

5.3 Semi-Markov Model

Let us establish the Semi-Markov model for the composite service C, where both services S_1 and S are protected by SCIT. The composite service C is denoted by C = (S_1, S). Note that according to the architecture, S1 and S may not belong to the same container, since each container only aggregates services with the same level of QoS.

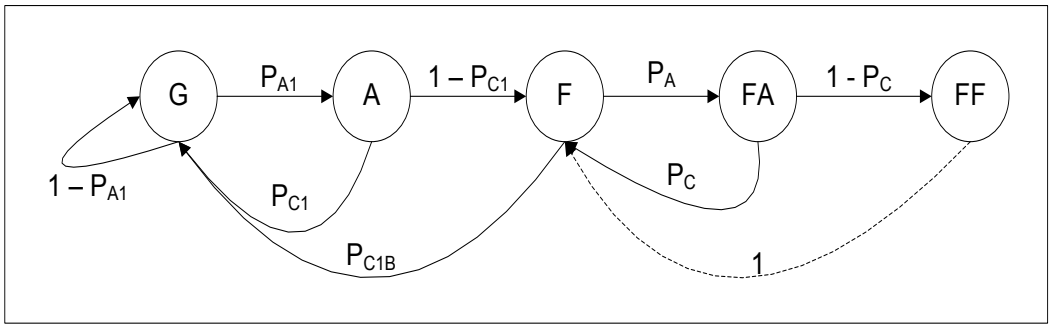

Figure 5.2. Transition Diagram of Composite Service

The transition diagram for C is depicted in Figure 5.2:

- G (Good): Services S_1 and S start with a Good state in which it functions normally.

- A (Attacked): As vulnerabilities are exploited, Service S_1 is under attack.
- F (Fail): If the attack is successful, then Service S_1 falls into Failure mode.
- FA: After S_1 is compromised, the attack continues onto Service S.
- FF: With S_1 being compromised, S has security failure.

Thus, the state space contains the following elements:

$$E = \{G, A, F, FA, FF\}$$

In terms of transition probabilities, the transition diagram shows:

- P_{A1} as the probability that an attack occurs so that service S_1 goes from state G to state A.
- P_{C1} as the probability of that the online service instance being under attack goes to cleansing mode, and is replaced by a pristine service instance. This makes service S_1 recovers to state G from state A.
- P_A as the probability that an attack occurs service S so that the system goes from state F to state FA.
- P_C as the probability of that the online service instance of S being under attack goes to cleansing mode, and is replaced by a pristine service instance. This makes service S go back to good state, so that the whole system is in state F. It is state F instead of G, because service S_1 is still in failure mode.

Note especially the transitions out of state F. From state F, the composite service can go back to state G thanks to the cleansing cycle of service S_1 with probability P_{C1B}. Hence, I can have a relationship between P_A and P_{C1B}:

$$P_{C1B} = 1 - P_A = \overline{P}_A \qquad (5.1).$$

Interestingly, (5.1) implies that if I increase the cleansing probability P_{C1B} of service S_1, then the attack probability P_A will be decreased. In other words, I can control (increase or decrease) the resilience of service S by tuning the cleansing probability of the enabler service S_1.

For the composite service C, the transition probability matrix **P** can be written as:

$$\boldsymbol{P} = \begin{array}{c} \\ G \\ A \\ F \\ FA \\ FF \end{array} \begin{array}{c} \begin{array}{ccccc} G & A & F & FA & FF \end{array} \\ \begin{pmatrix} \overline{P_{A1}} & P_{A1} & 0 & 0 & 0 \\ P_{C1} & 0 & \overline{P_{C1}} & 0 & 0 \\ \overline{P_A} & 0 & 0 & P_A & 0 \\ 0 & 0 & P_C & 0 & \overline{P_C} \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array},$$

where $\overline{P_{A1}} = 1 - P_A$, $\overline{P_{C1}} = 1 - P_C$, $\overline{P_A} = 1 - P_A$ and $\overline{P_C} = 1 - P_C$.

5.4 MTTSF Analysis

The analysis will use the same method performed for the case of an atomic service in the previous chapter; that is the behavior of the composite service with S_1 and S will be modeled using Semi-Markov process. The result of the analysis will help me obtaining the expressions of steady-state probabilities necessary to compute certain IT-QoS parameters.

5.4.1 MTTSF Expression

In order to compute the Mean Time to Security Failure (MTTSF) for service S (not S_1), the state FF is considered as an absorbing one. Within this context, the state pace E is divided into the set of transient states E_T and the set of absorbing states E_A: E_T = {G, A, F, FA} and E_A = {FF}. With these sets E_T and E_A, the transition probability matrix **Q** becomes:

$$\boldsymbol{Q} = \begin{array}{c} \\ G \\ A \\ F \\ FA \end{array} \begin{array}{c} \begin{array}{cccc} G & A & F & FA \end{array} \\ \begin{pmatrix} \overline{\mathrm{P_{A1}}} & \mathrm{P_{A1}} & 0 & 0 \\ \mathrm{P_{C1}} & 0 & \overline{\mathrm{P_{C1}}} & 0 \\ \overline{\mathrm{P_A}} & 0 & 0 & \mathrm{P_A} \\ 0 & 0 & \mathrm{P_C} & 0 \end{pmatrix} \end{array}.$$

Let:

- $\mathbf{X} = (\mathrm{X_G}, \mathrm{X_A}, \mathrm{X_F}, \mathrm{X_{FA}})$ be the row vector containing the number of visits at the transient in $\mathrm{X_T}$;
- $\mathbf{H} = (\mathrm{H_G}, \mathrm{H_A}\ \mathrm{H_F}, \mathrm{H_{FA}})$ the row vector containing the mean sojourn times at the transient in $\mathrm{X_T}$;
- $\mathbf{X_0} = (1, 0, 0, 0)$ being the starting vector.

Then, the number of visits X can be found by solving the system of equations using the methodology described in [Lim2001, Ma2004]:

$$\mathbf{X} = \mathbf{X_0} + \mathbf{XQ} \qquad (5.2)$$

Equation (5.2) can be expanded as:

$$(\mathrm{X_G}\ \mathrm{X_A}\ \mathrm{X_F}\ \mathrm{X_{FA}}) = (1\ \ 0\ \ 0\ \ 0) + (\mathrm{X_G}\ \mathrm{X_A}\ \mathrm{X_F}\ \mathrm{X_{FA}}) \begin{pmatrix} \overline{\mathrm{P_{A1}}} & \mathrm{P_{A1}} & 0 & 0 \\ \mathrm{P_{C1}} & 0 & \overline{\mathrm{P_{C1}}} & 0 \\ \overline{\mathrm{P_A}} & 0 & 0 & \mathrm{P_A} \\ 0 & 0 & \mathrm{P_C} & 0 \end{pmatrix},$$

yielding the equations in terms of $\mathrm{X_G}$, $\mathrm{X_A}$, $\mathrm{X_F}$, and $\mathrm{X_{FA}}$:

$$(5.2) \Leftrightarrow \begin{cases} \mathrm{X_G} = 1 + \mathrm{X_G}\overline{\mathrm{P_{A1}}} + \mathrm{X_A}\mathrm{P_{C1}} + \mathrm{X_F}\overline{\mathrm{P_A}} \\ \mathrm{X_A} = \mathrm{X_G}\mathrm{P_{A1}} \\ \mathrm{X_F} = \mathrm{X_A}\overline{\mathrm{P_{C1}}} + \mathrm{X_{FA}}\mathrm{P_C} \\ \mathrm{X_{FA}} = \mathrm{X_F}\mathrm{P_A} \end{cases}$$

$$\Leftrightarrow \begin{cases} X_G = 1 + X_G\overline{P_{A1}} + X_A P_{C1} + X_F\overline{P_A} \\ X_A = X_G P_{A1} \\ X_F = X_A\overline{P_{C1}} + X_F P_A P_C \\ X_{FA} = X_F P_A \end{cases}$$

$$\Leftrightarrow \begin{cases} X_G = 1 + X_G\overline{P_{A1}} + X_A P_{C1} + X_F\overline{P_A} \\ X_A = X_G P_{A1} \\ X_F = \dfrac{X_A\overline{P_{C1}}}{\overline{P_{AC}}} \\ X_{FA} = X_F P_A \end{cases}$$

$$\Leftrightarrow \begin{cases} X_G = 1 + X_G\overline{P_{A1}} + X_G P_{A1} P_{C1} + \dfrac{X_G P_{A1}\overline{P_{C1}}}{\overline{P_{AC}}}\overline{P_A} \\ X_A = X_G P_{A1} \\ X_F = \dfrac{X_G P_{A1}\overline{P_{C1}}}{\overline{P_{AC}}} \\ X_{FA} = \dfrac{X_G P_{A1}\overline{P_{C1}} P_A}{\overline{P_{AC}}} \end{cases}$$

$$\Leftrightarrow \begin{cases} X_G = \dfrac{1}{1 - \overline{P_{A1}} - P_{A1}P_{C1} - \dfrac{P_{A1}\overline{P_{C1}}}{\overline{P_{AC}}}\overline{P_A}} \\ X_A = X_G P_{A1} \\ X_F = \dfrac{X_A\overline{P_{C1}}}{\overline{P_{AC}}} \\ X_{FA} = \dfrac{X_G P_{A1}\overline{P_{C1}} P_A}{\overline{P_{AC}}} \end{cases}$$

$$\Leftrightarrow \begin{cases} X_G = \dfrac{\overline{P_{AC}}}{P_{A1}\overline{P_{C1}}P_A\overline{P_C}} \\ X_A = \dfrac{\overline{P_{AC}}}{\overline{P_{C1}}P_A\overline{P_C}} \\ X_F = \dfrac{1}{P_A\overline{P_C}} \\ X_{FA} = \dfrac{1}{\overline{P_C}} \end{cases} \qquad (5.3),$$

where $\overline{P_{AC}} = 1 - P_A P_C$.

According to [Ros2009], MTTSF is given by the scalar product:

$$\text{MTTSF} = \mathbf{X} . \mathbf{H}$$

Then, plugging the solution for **X** listed in (5.3) gives us the following expression for MTTSF:

$$\text{MTTSF} = X_G H_G + X_A H_A + X_F H_F + X_{FA} H_{FA}$$

$$= X_G H_G + X_G P_{A1} H_A + X_G \frac{P_{A1}\overline{P_{C1}}}{\overline{P_{AC}}} H_F + X_G \frac{P_{A1}\overline{P_{C1}}P_A}{\overline{P_{AC}}} H_{FA}$$

$$= \frac{\overline{P_{AC}}}{P_{A1}\overline{P_{C1}}P_A\overline{P_C}} H_G + \frac{\overline{P_{AC}}}{\overline{P_{C1}}P_A\overline{P_C}} H_A + \frac{1}{P_A\overline{P_C}} H_F + \frac{1}{\overline{P_C}} H_{FA} \quad (5.4).$$

5.4.2 Variation of MTTSF

Proposition 5.1. Let C = {S_1, S} be a composite service, with S_1 being the enabler, and S the target. The Mean Time to Security Failure (MTTSF) of the target S depends on the attack probability P_{A1} and the cleansing probability P_{C1} of the enabler S_1, in such a way that decreasing P_{A1} and increasing P_{C1} will increase MTTSF of S.

This proposition can be proved by computing the partial derivatives of MTTSF given in (5.4) with respect to P_{A1}, and P_{C1}:

$$\frac{\partial(\text{MTTSF})}{\partial P_{A1}} = -\frac{\overline{P_{AC}} H_G}{P_{A1}^2 \overline{P_{C1}} P_A P_C} < 0 \quad (5.5),$$

$$\frac{\partial(\mathrm{MTTSF})}{\partial \mathrm{P_{C1}}} = \frac{1}{(\overline{\mathrm{P_{C1}}})^2}\left(\frac{\overline{\mathrm{P_{AC}}}}{\mathrm{P_{A1}}\ \mathrm{P_A P_C}}\mathrm{H_G} + \frac{\overline{\mathrm{P_{AC}}}}{\mathrm{P_A}\overline{\mathrm{P_C}}}\mathrm{H_A}\right) > 0 \quad (5.6).$$

According to my study in the case of an atomic service, MTTSF increases when P_A decreases and P_C increases. For composite service, let study the variations of MTTSF with respect to P_A and P_C to make sure that it is still the case.

$$\frac{\partial(\mathrm{MTTSF})}{\partial \mathrm{P_A}} = -\frac{1}{\mathrm{P_A^2}}\left(\frac{\overline{\mathrm{P_{AC}}}}{\mathrm{P_{A1}}\overline{\mathrm{P_{C1}}}\overline{\mathrm{P_C}}}\mathrm{H_G} + \frac{\overline{\mathrm{P_{AC}}}}{\overline{\mathrm{P_{C1}}}\overline{\mathrm{P_C}}}\mathrm{H_A} + \frac{1}{\overline{\mathrm{P_C}}}\mathrm{H_F}\right) - \frac{\mathrm{P_C}}{\mathrm{P_A}}\left(\frac{1}{\mathrm{P_{A1}}\overline{\mathrm{P_{C1}}}\overline{\mathrm{P_C}}}\mathrm{H_G} + \frac{1}{\overline{\mathrm{P_{C1}}}\overline{\mathrm{P_C}}}\mathrm{H_A}\right)$$

$$< 0 \qquad (5.7)$$

$$\frac{\partial(\mathrm{MTTSF})}{\partial \mathrm{P_C}} = \frac{1}{(\overline{\mathrm{P_C}})^2}\left(\frac{\overline{\mathrm{P_{AC}}}}{\mathrm{P_{A1}}\overline{\mathrm{P_{C1}}}\mathrm{P_A}}\mathrm{H_G} + \frac{\overline{\mathrm{P_{AC}}}}{\overline{\mathrm{P_{C1}}}\mathrm{P_A}}\mathrm{H_A} + \frac{1}{\mathrm{P_A}}\mathrm{H_F} + \mathrm{H_{FA}}\right)$$

$$-\frac{\mathrm{P_A}}{\overline{\mathrm{P_C}}}\left(\frac{1}{\mathrm{P_{A1}}\overline{\mathrm{P_{C1}}}\overline{\mathrm{P_C}}}\mathrm{H_G} + \frac{1}{\overline{\mathrm{P_{C1}}}\overline{\mathrm{P_C}}}\mathrm{H_A}\right).$$

Grouping the terms involving H_G and H_A in the partial derivate of MTTSF with respect to P_C, I obtain:

$$\frac{\partial(\mathrm{MTTSF})}{\partial \mathrm{P_C}} = \frac{\overline{\mathrm{P_{AC}}} - \mathrm{P_A}\overline{\mathrm{P_C}}}{(\overline{\mathrm{P_C}})^2}\left(\frac{1}{\mathrm{P_{A1}}\overline{\mathrm{P_{C1}}}\mathrm{P_A}}\mathrm{H_G} + \frac{1}{\overline{\mathrm{P_{C1}}}\mathrm{P_A}}\mathrm{H_A}\right) + \frac{1}{(\overline{\mathrm{P_C}})^2}\left(\frac{1}{\mathrm{P_A}}\mathrm{H_F} + \mathrm{H_{FA}}\right)$$

$$\frac{\partial(\mathrm{MTTSF})}{\partial \mathrm{P_C}} = \frac{\overline{\mathrm{P_A}}}{(\overline{\mathrm{P_C}})^2}\left(\frac{1}{\mathrm{P_{A1}}\overline{\mathrm{P_{C1}}}\mathrm{P_A}}\mathrm{H_G} + \frac{1}{\overline{\mathrm{P_{C1}}}\mathrm{P_A}}\mathrm{H_A}\right) + \frac{1}{(\overline{\mathrm{P_C}})^2}\left(\frac{1}{\mathrm{P_A}}\mathrm{H_F} + \mathrm{H_{FA}}\right)$$

$$> 0 \qquad (5.8).$$

Regarding the variation of MTTSF with respect to P_{C1B}, I can write:

$$\frac{\partial(\mathrm{MTTSF})}{\partial \mathrm{P_{C1B}}} = \frac{\partial(\mathrm{MTTSF})}{\partial \mathrm{P_A}} \mathrm{x} \frac{\partial(\mathrm{P_A})}{\partial \mathrm{P_{C1B}}} = -\frac{\partial(\mathrm{MTTSF})}{\partial \mathrm{P_A}} > 0 \quad (5.9).$$

The inequalities (5.7) and (5.8) clearly show that the MTTSF of service S can be controlled by the attack probability P_A on S and its cleaning probability P_C, as in the case of an atomic service. In addition, (5.9) reveals that the cleansing probability P_{C1B} also impacts service S in a positive way, meaning that the MTTSF of S increases as P_{C1B} does.

5.5 S-Availability Analysis

The derivation of the expression for the S-Availability (SA) can be done using the same approach as in the atomic service. By not considering state F as an absorbing state [Ma2004], I will use the entire transition matrix **P** encompassing all states in E = {G, A, F, FA, FF}.

Let:

- **X** = (X_G X_A X_F X_{FA} X_{FF}) be the row vector containing the steady-state probabilities to be in states G, A and F in the embedded Markov chain respectively;
- **H** = (H_G H_A H_F H_{FA} H_{FF}) the row vector containing the mean sojourn times at the states in E.

The stead-state probabilities can be found by solving the system of equations:

$$\begin{cases} \mathbf{X} = \mathbf{X}.\mathbf{P} \\ \sum_{i \in E} X_i = 1 \end{cases} \qquad (5.10)\cdot$$

The matrix equation **X** = **X.P** in (5.10) can be expanded as:

$$\{ (X_G\ X_A\ X_F\ X_{FA}\ X_{FF}) = (X_G\ X_A\ X_F\ X_{FA}\ X_{FF}) \begin{pmatrix} \overline{P_{A1}} & P_{A1} & 0 & 0 & 0 \\ P_{C1} & 0 & \overline{P_{C1}} & 0 & 0 \\ \overline{P_A} & 0 & 0 & P_A & 0 \\ 0 & 0 & P_C & 0 & \overline{P_C} \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

$$(5.9) \Leftrightarrow \begin{cases} X_G = X_G\overline{P_{A1}} + X_A P_{C1} + X_F\overline{P_A} + X_{FF} \\ X_A = X_G P_{A1} \\ X_F = X_A\,\overline{P_{C1}} + X_{FA} P_C \\ X_{FA} = X_F P_A \\ X_{FF} = X_{FA}\overline{P_C} \\ X_G + X_A + X_F + X_{FA} + X_{FF} = 1 \end{cases}$$

$$\Leftrightarrow \begin{cases} X_G = X_G\overline{P_{A1}} + X_A P_{C1} + X_F\overline{P_A} + X_{FF} \\ X_A = X_G P_{A1} \\ X_F = \dfrac{X_G P_{A1}\overline{P_{C1}}}{\overline{P_{AC}}} \\ X_{FA} = \dfrac{X_G P_{A1}\overline{P_{C1}} P_A}{\overline{P_{AC}}} \\ X_{FF} = \dfrac{X_G P_{A1}\overline{P_{C1}} P_A\overline{P_C}}{\overline{P_{AC}}} \\ X_G + X_A + X_F + X_{FA} + X_{FF} = 1 \end{cases}$$

$$\Leftrightarrow \begin{cases} X_G = X_G\overline{P_{A1}} + X_A P_{C1} + X_F\overline{P_A} + X_{FF} \\ X_A = X_G P_{A1} \\ X_F = \dfrac{X_G P_{A1}\overline{P_{C1}}}{\overline{P_{AC}}} \\ X_{FA} = \dfrac{X_G P_{A1}\overline{P_{C1}} P_A}{\overline{P_{AC}}} \\ X_{FF} = \dfrac{X_G P_{A1}\overline{P_{C1}} P_A\overline{P_C}}{\overline{P_{AC}}} \\ X_G + X_G P_{A1} + \dfrac{X_G P_{A1}\overline{P_{C1}}}{\overline{P_{AC}}} + \dfrac{X_G P_{A1}\overline{P_{C1}} P_A}{\overline{P_{AC}}} + \dfrac{X_G P_{A1}\overline{P_{C1}} P_A\overline{P_C}}{\overline{P_{AC}}} = 1 \end{cases}$$

This yields the expression for **X**:

$$\Leftrightarrow \begin{cases} X_A = X_G P_{A1} \\ X_F = \dfrac{X_G P_{A1} \overline{P_{C1}}}{\overline{P_{AC}}} \\ X_{FA} = \dfrac{X_G P_{A1} \overline{P_{C1}} P_A}{\overline{P_{AC}}} \\ X_{FF} = \dfrac{X_G P_{A1} \overline{P_{C1}} P_A \overline{P_C}}{\overline{P_{AC}}} \\ X_G = {\Delta_2}^{-1} \end{cases} \quad (5.11),$$

where $\Delta_2 = 1 + P_{A1} + \frac{P_{A1}\overline{P_{C1}}}{\overline{P_{AC}}} + \frac{P_{A1}\overline{P_{C1}}P_A}{\overline{P_{AC}}} + \frac{P_{A1}\overline{P_{C1}}P_A\overline{P_C}}{\overline{P_{AC}}}$.

Since SA of service S within the composite service is the time not spent in state FF, SA can be given by the formula [Ros2009, Tri2002, Ma2004]:

$$SA = 1 - \frac{X_{FF} H_{FF}}{\mathbf{X}.\mathbf{H}}.$$

Plugging the solution for **X** expressed in (5.10) into (5.11) yields the following expression for SA:

$$SA = 1 - \frac{P_{A1}\overline{P_{C1}}\, P_A \overline{P_C}\, H_{FF}}{(H_G + H_A P_{A1})(\overline{P_{AC}}) + P_{A1}\overline{P_{C1}}\,[H_F + P_A(H_{FA} + H_{FF}\overline{P_C})]} \quad (5.12),$$

Let $\Delta_3 = (H_G + H_A P_{A1})(\overline{P_{AC}}) + P_{A1}\overline{P_{C1}}\,[H_F + P_A(H_{FA} + H_{FF}\overline{P_C})]$.

Proposition 5.2. Let C = {S_1, S} be a composite service, with S_1 being the enabler, and S the target. The S-Availability (SA) of the target S depends on the attack probability P_{A1} and the cleansing probability P_{C1B} of the enabler S_1, in such a way that decreasing P_{A1} and increasing P_{C1}, P_{C1B} will increase SA of S.

Similarly to MTTSF, Proposition 5.2 can be proven by computing the partial derivatives of SA given in (5.12) with respect to P_{A1} and P_{C1}.

Compute the partial derivative of Δ_3 in (5.12) with respect to P_{A1}:

$$\frac{\partial(\Delta_3)}{\partial P_{A1}} = H_A(\overline{P_{AC}}) + \overline{P_{C1}}\,[H_F + P_A(H_{FA} + H_{FF}\overline{P_C})].$$

Using the partial derivative of Δ_3 in the partial derivative of SA with respect to P_{A1}, I obtain:

$$\frac{\partial(SA)}{\partial P_{A1}} = -\frac{\overline{P_{C1}\,P_C}\,P_A H_{FF}}{(\Delta_3)^2}(\Delta_3 - P_{A1}(H_A(\overline{P_{AC}}) + \overline{P_{C1}}\,[H_F + P_A(H_{FA} + H_{FF}\overline{P_C})])$$

$$\frac{\partial(SA)}{\partial P_{A1}} = -\frac{\overline{P_{C1}\,P_C}\,P_A H_{FF}}{(\Delta_3)^2}(H_G)(\overline{P_{AC}}) < 0 \quad (5.13).$$

Compute the partial derivative of Δ_3 in (5.12) with respect to P_{C1}:

$$\frac{\partial(\Delta_3)}{\partial P_{C1}} = -P_{A1}[H_F + P_A(H_{FA} + H_{FF}\overline{P_C})].$$

Using the partial derivative of Δ_3 in the partial derivative of SA with respect to P_{A1}, I obtain:

$$\frac{\partial(SA)}{\partial P_{C1}} = -\frac{P_{A1}P_A\,\overline{P_C}\,H_{FF}}{(\Delta_3)^2}(-\Delta_3 + \overline{P_{C1}}\,(P_{A1}[H_F + P_A(H_{FA} + H_{FF}\overline{P_C})])$$

$$\frac{\partial(SA)}{\partial P_{C1}} = \frac{P_{A1}P_A\,\overline{P_C}\,H_{FF}}{(\Delta_3)^2}(H_G + H_A P_{A1})(\overline{P_{AC}}) > 0 \quad (5.14).$$

According to my study in the case of an atomic service, SA increases when P_A decreases and P_C increases. For a composite service, let study the variations of SA with respect to P_A and P_C to make sure that it is still the case.

Compute the partial derivative of Δ_3 in (5.12) with respect to P_A:

$$\frac{\partial(\Delta_3)}{\partial P_A} = -(H_G + H_A P_{A1})P_C + P_{A1}\overline{P_{C1}}\left[\left(H_{FA} + H_{FF}\overline{P_C}\right)\right].$$

Using the partial derivative of Δ_3 in the partial derivative of SA with respect to P_A, I obtain:

$$\frac{\partial(SA)}{\partial P_A} = -\frac{P_{A1}\overline{P_{C1}}\,\overline{P_C}H_{FF}}{(\Delta_3)^2}(\Delta_3 - P_A(-(H_G + H_A P_{A1}) + P_{A1}\overline{P_{C1}}\left[\left(H_{FA} + H_{FF}\overline{P_C}\right)\right])$$

$$\frac{\partial(SA)}{\partial P_A} = -\frac{P_{A1}\overline{P_{C1}}\,\overline{P_C}H_{FF}}{(\Delta_3)^2}\left[(H_G + H_A P_{A1})(\overline{P_{AC}}) + P_{A1}\overline{P_{C1}}\,H_F + P_A(H_G + H_A P_{A1})\right]$$

$$< 0 \quad (5.15).$$

From (5.15), I can also deduce that:

$$\frac{\partial(SA)}{\partial P_{C1B}} > 0.$$

Compute the partial derivative of Δ_3 in (5.12) with respect to P_C:

$$\frac{\partial(\Delta_3)}{\partial P_C} = -[(H_G + H_A P_{A1})P_A + P_{A1}\overline{P_{C1}}\,P_A H_{FF}].$$

Using the partial derivative of Δ_3 in the partial derivative of SA with respect to P_C, I obtain:

$$\frac{\partial(SA)}{\partial P_C} = -\frac{P_{A1}\overline{P_{C1}}\,P_A H_{FF}}{(\Delta_3)^2}(-\Delta_3 + \overline{P_C}\left((H_G + H_A P_{A1})P_A + P_{A1}\overline{P_{C1}}\,P_A H_{FF}\right))$$

$$\frac{\partial(SA)}{\partial P_C} = -\frac{P_{A1}\overline{P_{C1}}\,P_A H_{FF}}{(\Delta_3)^2}\{-((H_G + H_A P_{A1})(\overline{P_{AC}}) + P_{A1}\overline{P_{C1}}\,[H_F + P_A H_{FA}])$$

$$+ \overline{P_C}(H_G + H_A P_{A1})\,P_A\}$$

$$\frac{\partial(SA)}{\partial P_C} = -\frac{P_{A1}\overline{P_{C1}}\,P_A H_{FF}}{(\Delta_3)^2}\left[(H_G + H_A P_{A1})(-\overline{P_{AC}} + \overline{P_C}\,P_A) - P_{A1}\overline{P_{C1}}\,(H_F + P_A H_{FA})\right]$$

$$\frac{\partial(SA)}{\partial P_C} = \frac{P_{A1}\overline{P_{C1}}\,P_A H_{FF}}{(\Delta_3)^2}\left[(H_G + H_A P_{A1})(\overline{P_A}) + P_{A1}\overline{P_{C1}}\,(H_F + P_A H_{FA})\right] > 0 \quad (5.16).$$

From the results in (5.15) and (5.16), I can infer that SA of service S can be controlled by the attack probability P_A on S and its cleaning probability P_C, as in the case of an atomic service.

5.6 Other IT-QoS

For other IT-QoS, the study is the same as in the case of an atomic service. So, I will not repeat it here.

5.7 Independent Services

In this section, I consider the case of composite service C comprising of services S_1, S_2, S_3, etc. which are independent in terms of being attacked. Unlike the case of enabler-target composite, the order with which these services can be attacked does not depend on the sequence within the service chain.

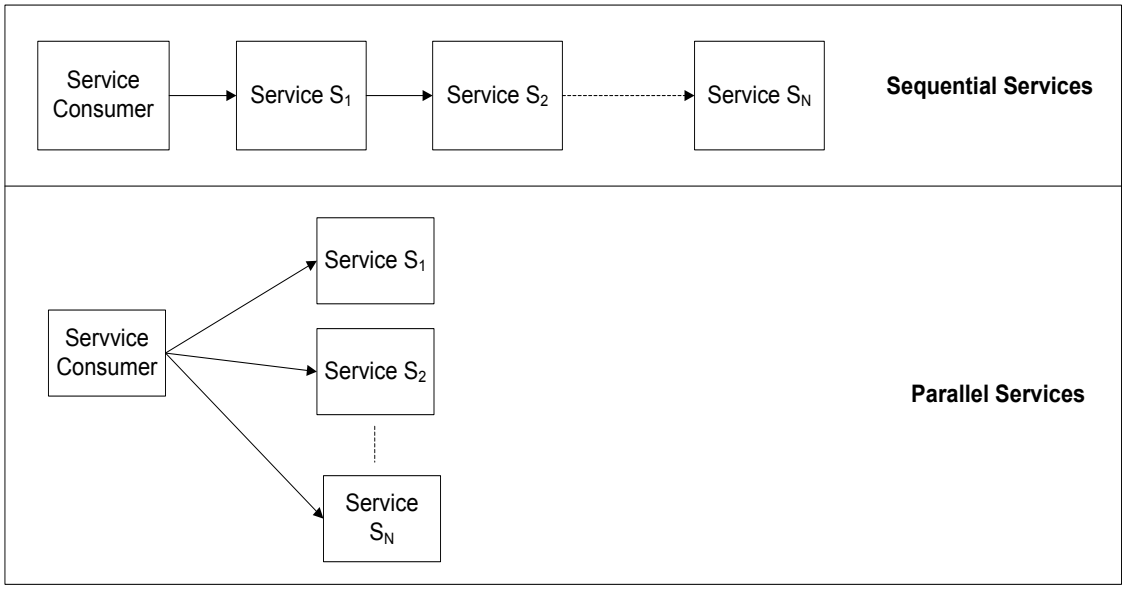

Figure 5.3. Composite of Independent Services

5.7.1 Transition Diagram

The paths formed by the transitions $G_1 \rightarrow G_2 \rightarrow A_2$ and $F_1 \rightarrow G_2$ in Fig. 5.4 mean that services S_1 and S_2 are independent when it comes to being attacked. The states of the composite C are:

- G1 (Good): Service S_1 starts with a Good state in which it functions normally.
- A1 (Attacked): As vulnerabilities are exploited, Service S_1 is under attack.
- F1 (Fail): If the attack is successful, then Service S_1 falls into Failure mode.
- G2: After S_1 is compromised, the attack continues onto Service S_2.
- A2 (Attacked): As vulnerabilities are exploited, Service S_2 is under attack.
- F2: If the attack is successful, then Service S_1 falls into Failure mode.

Thus, the state space contains the following elements:

$$E = \{G1, A1, F1, G2, A2, F2, G3, \ldots\}$$

For each service S_i (i=1, 2, 3, ...), P_{Ai} and P_{Ci} are respectively the probabilities that an attack occurs to service S_i and the probability that the online service instance of service SI goes to cleansing mode.

For the composite service C, the transition probability matrix **P** can be written as:

$$\boldsymbol{P} = \begin{array}{c} \\ G1 \\ A1 \\ F1 \\ G2 \\ A2 \\ F2 \\ G3 \end{array} \begin{array}{c} \begin{array}{ccccccc} G1 & A1 & F1 & G2 & A2 & F2 & G3 \end{array} \\ \begin{pmatrix} 0 & \mathrm{P_{A1}} & 0 & \overline{\mathrm{P_{A1}}} & 0 & 0 & . \\ \mathrm{P_{C1}} & 0 & \overline{\mathrm{P_{C1}}} & 0 & 0 & 0 & . \\ \mathrm{P_{C1B}} & 0 & 0 & \overline{\mathrm{P_{C1B}}} & 0 & 0 & . \\ 0 & 0 & 0 & 0 & \mathrm{P_{A2}} & 0 & \overline{\mathrm{P_{A2}}} \\ 0 & 0 & 0 & \mathrm{P_{C2}} & 0 & \overline{\mathrm{P_{C2}}} & . \\ 0 & 0 & 0 & \mathrm{P_{C2B}} & 0 & 0 & \overline{\mathrm{P_{C2B}}} \\ . & . & . & . & . & . & . \end{pmatrix} \end{array},$$

where $\overline{\mathrm{P_{A1}}} = 1 - \mathrm{P_A}$, $\overline{\mathrm{P_{C1}}} = 1 - \mathrm{P_C}$, $\overline{\mathrm{P_{A2}}} = 1 - \mathrm{P_{A2}}$ and $\overline{\mathrm{P_{C2}}} = 1 - \mathrm{P_{C2}}$.

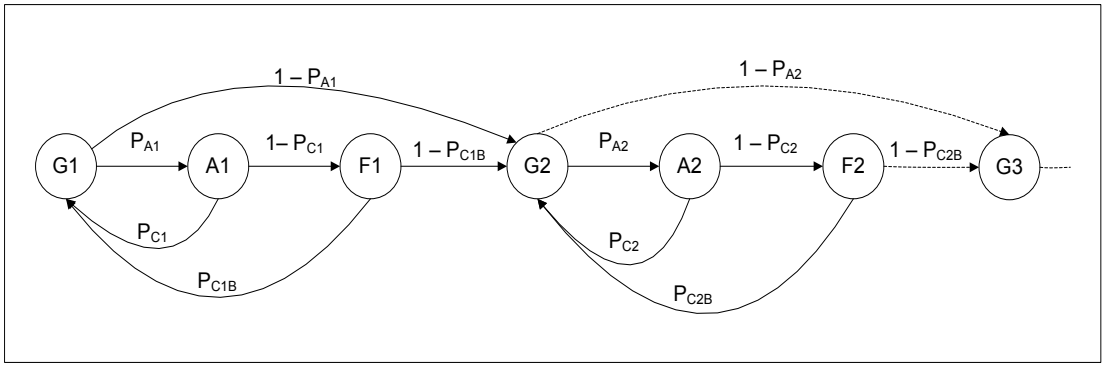

Figure 5.4. Composite of Independent Services

Since the services in the composite are independent, I can apply the methodology used in the previous chapter to analyze each service within the composite. The same conclusion that the exposure window is a parameter allowing the tuning of the IT-QoS such as MTTSF, MTTSR, S-Availability and S-Reliability should be reached for each service. Then, by applying Reliability Theory [Ros2009, Pha2006] for sequential and parallel services, I can derive the S-Reliability of the composite service C. Let SR(C) denote the S-Reliability of C, and $SR(S_i)$, i = 1, 2, 3, ..n be the S-Reliability of services S_i. According to [Ros2009, Pha2006], the S-Reliability of the composite service C is given by:

$$\begin{cases} SR(C) = \prod_{i=1}^{n} SR(S_i) & if\ S_i\ are\ sequential \\ SR(C) = \left[1 - \prod_{i=1}^{n} \left(1 - SR(S_i)\right)\right] & if\ S_i\ are\ parallel \end{cases}.$$

5.8 Summary

In fact, the utilization of the methods in previous sections gives inconclusive results as to the impact of P_{A1} and P_{C1} of Service S_1 onto the MTTSF and SA of Service S_2. However, each service S_1 and S_2 can be analyzed separately using the methodology for an Atomic Service above, and shown to be impacted by their own (P_{A1}, P_{C1}) and (P_{A2}, P_{C2}) respectively. Parallel and conditional branches in a service orchestration will behave similarly from the perspective of impact of cleansing and attack probabilities; hence, analyzing these cases is also tantamount to analyzing two separate atomic services.

5.9 Applications

5.9.1 Multi-tier Architecture

In a multi-tier architecture, services may require different IT-QoS levels depending on the layers where they are located per design in the architecture. For instance, data and storage services are important in an open archival system, thus will likely need high S-Reliability to secure and protect the digital assets. In [Ng2010a], the vertical QoS Layer has the role of managing QoS, including IT-QoS. The following figure from [Ng2010a] illustrates the design of an open digital archival system as specified in [Con2002], using a multi-tier architecture protected by SCIT. The three layers in the architecture are, from top to bottom:

- Presentation layer with the access portal;
- Business Layer with the service named AccessDigitalRecord to manage the access to digital records in the archive.
- Utility Layer with three core services for managing the records in storage (StoreDigitalRecords), virus scanning the records (VirusScan) before presenting the records to the client, and build the dissemination package (BuildDisseminationPackage).

SCIT Central Controller and its cleansing algorithms fit into the vertical QoS management layer and controls the IT-QoS of the services in the horizontal layers.

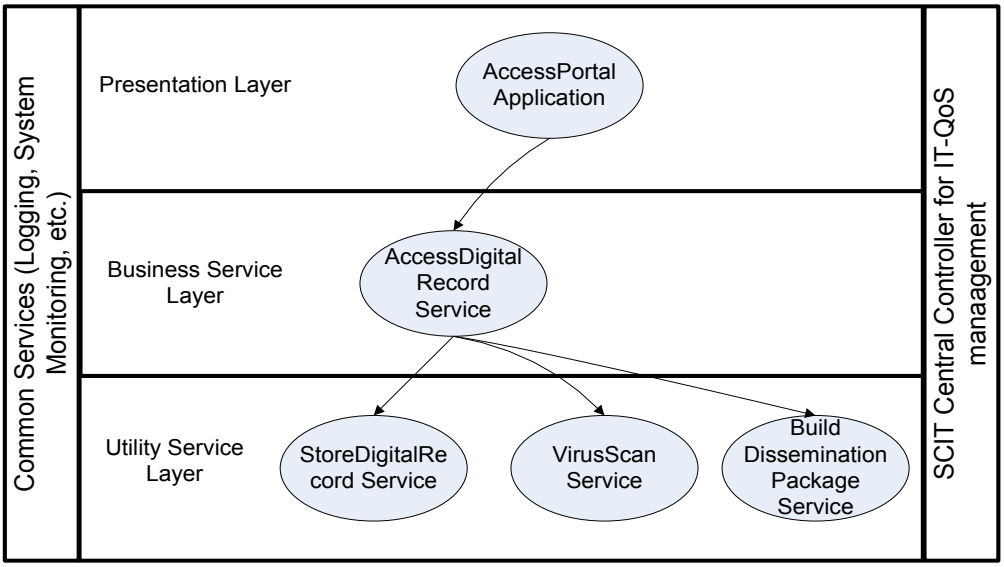

Figure 5.5. SCIT and Multi-tier Architecture [Ng2010a].

5.9.2 SOA Ecosystem

With today's proliferation of web services, an application can be constructed out of services from different providers. For instance, Amazon S3 storage service, Google map service, and a business rule service can be orchestrated to form an application. In [Ng2013], the authors propose a resilient and intrusion tolerant SOA ecosystem, in which service providers have their operational environments protected by SCIT. Each service provider will have its own Central Controller to manage the cleansing of its service containers. Thus, a service provider will be able to control and quantitatively tune its published IT-QoS via the control parameters. As a result, an application can select a service provider for a service that fits the desired level of IT-QoS for the service composition to be used within the application.

CHAPTER 6 - COMBINATION OF SCIT AND IDS

6.1 Overview

As discussed in chapter 2, an Intrusion Tolerance System (ITS) can be based on one strategy or a combination of strategies, such as recovery-based mechanism, intrusion detection, secret sharing, etc. Using the same methodology based on Semi-Markov Process, I will analyze the behavior of a service S protected by the combination of two components: SCIT and intrusion detection, denoted by (SCIT+IDS). The cost point of view of this integrated strategy was studied in [Nag2011] using Bayes probability formula. The outcome of the analysis is to obtain the expressions for MTTSF and S-Availability, from which variation analysis will be done using differentiation calculus. In this chapter, I will consider Intrusion Detection System (IDS) at the abstract level without delving into the specific algorithms underlying the detection mechanism. In other words, instead of going to the level of what intrusion mechanisms and algorithms being used, I will characterize the intrusion detection component by its false positive probability and detection probability [Axe2000, Gaf2001]. Therefore, with (SCIT+IDS), the control parameters will comprise: the exposure window, the detection probability and the false positive probability. From the expressions obtained via the Semi-Markov Process computation, I will establish the impact of these control parameters on the selected IT-QoS, according to Module 3 of QFITS.

6.2 Intrusion Detection

At the high level, an intrusion detection system (IDS) utilizes signature-based scheme, anomaly-based scheme, or combination of both [Axe2000, Gaf2001]. Within each algorithm, an IDS typically makes use of other information technology techniques to realize or enhance these two main schemes.

A *signature-based IDS* maintains an internal database of attack signatures in order to match an incoming attack with one of the existing signatures for detection malicious intrusion. The signature database has to be maintained to incorporate the latest attack signatures. System procedures have to be instilled to perform updates as quickly as possible. Thus, by its nature, the drawback of this category of IDS is its inefficacity to detect novel lines of attacks or zero-day attacks.

An *anomaly-based IDS* may be able to detect new lines of attacks. Instead of maintaining a signature database, this category of IDS learns the normal behaviors of users and systems being in operation. The normal behavior can also be specified by a set of policies. For example, a rule limiting the size of a download can be configured in the IDS knowledge base. On the other hand, machine learning techniques can be implemented in the IDS to observe and learn the normal usage patterns of the users. Although providing the capability to detect new malicious activities, the effectiveness of an anomaly-based IDS depends on the extent of its knowledge base and predictive adjudication of observed system and user activities.

6.2.1 Control Parameters

At the abstract level, the control parameters of IDS are, according to [Axe2000, Gaf2001]:

- Detection Probability, which is the probability to output an alarm when there is an intrusion;
- False Positive Probability, which is the probability of outputting an alarm when where there is no intrusion.

Therefore a service protected by IDS will have two control parameters, namely the detection probability P_D, and the false positive probability P_{FP}.

6.2.2 Semi-Markov Model

The transition diagram for IDS is depicted in Figure 6.1:

- G (Good): Service S starts with a Good state in which it functions normally.
- A (Attacked): As vulnerabilities are exploited, Service S is under attack.
- FP (False Positive): With the IDS in place, a false alert may be triggered, causing the online service instance to go to cleansing and be replaced by a fresh instance. Hence, the service can be in Good state again.
- D (Detected): With the IDS in place, the attack in state A can be detected, triggering online service instance to go to cleansing and be replaced by a fresh instance. Hence, the service can be in Good state again.
- F (Fail): If the attack is not detected and cleansing does not happen in time, then Service S will fall into Failure mode.

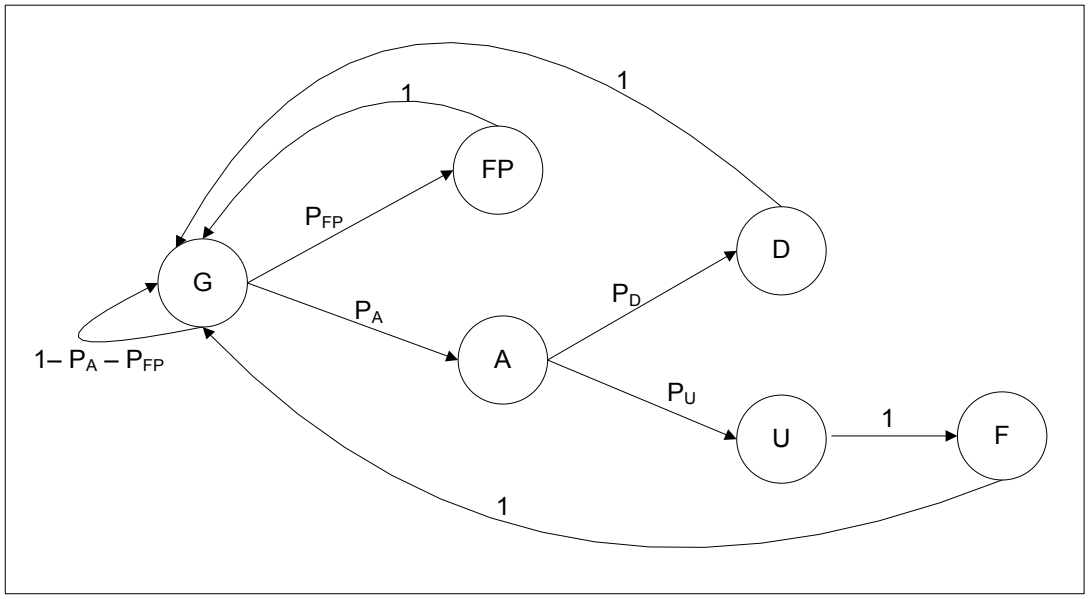

Figure 6.1. Service Protected by IDS

Thus, the state space contains the following elements:

$$E = \{G, A, FP, D, U, F\}$$

In terms of transition probabilities, the transition diagram shows:

- P_A as the probability that an attack occurs so that service S goes from state G to state A.
- P_{FP} as the probability of a false positive, which makes the service go to state FP, then to cleansing mode.
- P_D as the probability of detection, which makes the service go from state A to state D.
- P_U as the probability of detection miss . Thus, the transition from state A to state U has probability P_U.

Thus, the transition probability matrix **P** can be written as:

$$\boldsymbol{P} = \begin{array}{c} \\ G \\ A \\ FP \\ D \\ U \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} G & A & FP & D & U & F \end{array} \\ \begin{pmatrix} \overline{\mathrm{P_{AFP}}} & \mathrm{P_A} & \mathrm{P_{FP}} & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathrm{P_D} & \mathrm{P_U} & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array},$$

where $\overline{\mathrm{P_{AFP}}} = 1 - \mathrm{P_A} - \mathrm{P_{FP}}$ and $\mathrm{P_U} = 1 - \mathrm{P_D}$.

6.3 MTTSF Analysis

In this section, I will derive the expression of MTTSF in terms of the probabilities of attack, detection and false positive, and study how MTTSF vary with respect to these probabilities.

6.3.1 MTTSF Expression

In order to compute the Mean Time to Security Failure (MTTSF) for service S, the state F is considered as an absorbing one.

Within this context, the state pace E is divided into the set of transient states $\mathrm{E_T}$ and the set of absorbing states $\mathrm{E_A}$:

$\mathrm{E_T}$ = {G, A, FP, D, U} and

$\mathrm{E_A}$ = {F}.

With these sets $\mathrm{E_T}$ and $\mathrm{E_A}$, the transition probability matrix **Q** becomes:

$$\begin{array}{ccccc} G & A & FP & D & U \end{array}$$

$$\boldsymbol{Q} = \begin{matrix} G \\ A \\ FP \\ D \\ U \end{matrix} \begin{pmatrix} \overline{\mathrm{P_{AFP}}} & \mathrm{P_A} & \mathrm{P_{FP}} & 0 & 0 \\ 0 & 0 & 0 & \mathrm{P_D} & \mathrm{P_U} \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Let:

- $\mathbf{X} = (\mathrm{X_G}\ \mathrm{X_A}\ \mathrm{X_{FP}}\ \mathrm{X_D}\ \mathrm{X_U})$ be the row vector containing the number of visits at the transient in $\mathrm{E_T}$;
- $\mathbf{H} = (\mathrm{H_G}\ \mathrm{H_A}\ \mathrm{H_{FP}}\ \mathrm{H_D}\ \mathrm{H_U})$ the row vector containing the mean sojourn times at the transient in $\mathrm{E_T}$;
- $\mathbf{X_0} = (1\ 0\ 0\ 0\ 0)$ being the starting vector.

Then, the number of visits X can be found by solving the system of equations using the methodology described in [Lim2001, Ma2004]:

$$\mathbf{X} = \mathbf{X_0} + \mathbf{XQ} \qquad (6.1)$$

Equation (6.1) can be expanded as:

$$(\mathrm{X_G}\ \mathrm{X_A}\ \mathrm{X_{FP}}\ \mathrm{X_D}\ \mathrm{X_U})$$

$$= (1\ 0\ 0\ 0\ 0) + (\mathrm{X_G}\ \mathrm{X_A}\ \mathrm{X_{FP}}\ \mathrm{X_D}\ \mathrm{X_U}) \begin{pmatrix} \overline{\mathrm{P_{AFP}}} & \mathrm{P_A} & \mathrm{P_{FP}} & 0 & 0 \\ 0 & 0 & 0 & \mathrm{P_D} & \mathrm{P_U} \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix},$$

yielding the equations in terms of $\mathrm{X_G}$, $\mathrm{X_A}$, $\mathrm{X_{FP}}$, $\mathrm{X_D}$ and $\mathrm{X_U}$:

$$(6.1) \Leftrightarrow \begin{cases} \mathrm{X_G} = 1 + \mathrm{X_G}\overline{\mathrm{P_{AFP}}} + \mathrm{X_{FP}} + \mathrm{X_D} \\ \mathrm{X_A} = \mathrm{X_G}\mathrm{P_A} \\ \mathrm{X_{FP}} = \mathrm{X_G}\mathrm{P_{FP}} \\ \mathrm{X_D} = \mathrm{X_A}\mathrm{P_D} \\ \mathrm{X_U} = \mathrm{X_A}\mathrm{P_U} \end{cases}$$

$$\Leftrightarrow \begin{cases} X_G = 1 + X_G\overline{P_{AFP}} + X_{FP} + X_D \\ X_A = X_G P_A \\ X_{FP} = X_G P_{FP} \\ X_D = X_G P_A P_D \\ X_U = X_G P_A P_U \end{cases}$$

$$\Leftrightarrow \begin{cases} X_G = 1 + X_G\overline{P_{AFP}} + X_G P_{FP} + X_G P_A P_D \\ X_A = X_G P_A \\ X_{FP} = X_G P_{FP} \\ X_D = X_G P_A P_D \\ X_U = X_G P_A P_U \end{cases}$$

$$\Leftrightarrow \begin{cases} X_G = \dfrac{1}{\Delta_6} \\ X_A = X_G P_A \\ X_{FP} = X_G P_{FP} \\ X_D = X_G P_A P_D \\ X_U = X_G P_A P_U \end{cases} \qquad (6.2),$$

where $\Delta_6 = 1 - \overline{P_{AFP}} - P_{FP} - P_A P_D = P_A - P_A P_D = P_A P_U$.

According to [Ros2009, Ma2004], MTTSF is given by the scalar product:

$$\text{MTTSF} = \mathbf{X}.\,\mathbf{H}$$

Then, plugging the solution for **X** listed in (6.2) gives us the following expression for MTTSF:

$$\text{MTTSF} = X_G H_G + X_A H_A + X_{FP} H_{FP} + X_D H_D + X_U H_U$$

$$\text{MTTSF} = \frac{H_G + P_A H_A + P_{FP} H_{FP} + P_A P_D H_D + P_A P_U H_U}{P_A P_U} \qquad (6.3)$$

6.3.2 Variation of MTTSF

Computing the partial derivatives of MTTSF in (6.3) with respect to P_A, I got:

$$\frac{\partial(\mathrm{MTTSF})}{\partial \mathrm{P_A}}$$

$$= \frac{(\mathrm{H_A} + \mathrm{P_D H_D} + \mathrm{P_U H_U})\mathrm{P_A P_U} - (\mathrm{H_G} + \mathrm{P_A H_A} + \mathrm{P_{FP} H_{FP}} + \mathrm{P_A P_D H_D} + \mathrm{P_A P_U H_U})\mathrm{P_U}}{(\mathrm{P_A P_U})^2}$$

$$\frac{\partial(\mathrm{MTTSF})}{\partial \mathrm{P_A}} = \frac{-(\mathrm{H_G} + \mathrm{P_{FP} H_{FP}})\mathrm{P_U}}{(\mathrm{P_A P_U})^2} < 0 \qquad (6.4).$$

The partial derivative of MTTSF with respect to P_{FP} yields:

$$\frac{\partial(\mathrm{MTTSF})}{\partial \mathrm{P_{FP}}} = \frac{\mathrm{P_{FP} H_{FP}}}{\mathrm{P_A P_U}} > 0 \qquad (6.5).$$

The partial derivative of MTTSF with respect to P_D yields:

$$\frac{\partial(\mathrm{MTTSF})}{\partial \mathrm{P_D}} = \frac{(\mathrm{P_A H_D} - \mathrm{P_A H_U})\mathrm{P_A P_U} + (\mathrm{H_G} + \mathrm{P_A H_A} + \mathrm{P_{FP} H_{FP}} + \mathrm{P_A P_D H_D} + \mathrm{P_A P_U H_U})\mathrm{P_A}}{(\mathrm{P_A P_U})^2}$$

$$\frac{\partial(\mathrm{MTTSF})}{\partial \mathrm{P_D}} = \frac{\mathrm{P_A P_U H_D} + \mathrm{H_G} + \mathrm{P_A H_A} + \mathrm{P_{FP} H_{FP}} + \mathrm{P_A P_D H_D}}{\mathrm{P_A P_U}^2} > 0 \qquad (6.6).$$

From all these four partial derivatives in (6.4)-(6.6), I can infer that MTTSF of service S will increase when the probability of alerts increases; these alerts can be real detection of intrusions with P_D, or false positives with P_{FP}.

6.4 S-Availability Analysis

In this section, I will derive the expression of S-Availability (SA) in terms of the probabilities of attack, detection and false positive, and study how SA vary with respect to these probabilities.

6.4.1 S-Availability Expression

The derivation of the expression for the SA can be done using the same approach as in the atomic service. By not considering state F as an absorbing state, I will use the entire transition matrix **P** encompassing all states in E = {G, A, FP, D, U, F}.

Let:

- **X** = (X_G X_A X_{FP} X_D X_U X_F) be the row vector containing the steady-state probabilities at the states in E of the embedded Markov chain;
- **H** = (H_G H_A H_{FP} H_D H_U H_F) the row vector containing the mean sojourn times at the states in E.

The steady-state probabilities can be solved from the system of equations:

$$\begin{cases} \mathbf{X} = \mathbf{XP} \\ \sum_{i \in E} X_i = 1 \end{cases} \qquad (6.7)\cdot$$

The matrix equation X = XP in the system (6.7) can be expanded as:

$$(X_G\ X_A\ X_{FP}\ X_D\ X_U\ X_F) = (X_G\ X_A\ X_{FP}\ X_D\ X_U\ X_F)\begin{pmatrix} \overline{P_{AFP}} & P_A & P_{FP} & 0 & 0 & 0 \\ 0 & 0 & 0 & P_D & P_U & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Then:

$$(6.7) \Leftrightarrow \begin{cases} X_G = X_G\overline{P_{AFP}} + X_{FP} + X_D + X_F \\ X_A = X_G P_A \\ X_{FP} = X_G P_{FP} \\ X_D = X_A P_D \\ X_U = X_A P_U \\ X_F = X_U \\ X_G + X_A + X_F + X_{FA} + X_{FF} = 1 \end{cases}$$

$$\Leftrightarrow \begin{cases} X_G = X_G\overline{P_{AFP}} + X_{FP} + X_D + X_F \\ X_A = X_G P_A \\ X_{FP} = X_G P_{FP} \\ X_D = X_G P_A P_D \\ X_U = X_G P_A P_U \\ X_F = X_G P_A P_U \\ X_G + X_G P_A + X_G P_{FP} + X_G P_A P_D + X_G P_A P_U + X_G P_A P_U = 1 \end{cases}$$

$$\Leftrightarrow \begin{cases} X_A = X_G P_A \\ X_{FP} = X_G P_{FP} \\ X_D = X_G P_A P_D \\ X_U = X_G P_A P_U \\ X_F = X_G P_A P_U \\ X_G = \dfrac{1}{1 + P_A + P_{FP} + P_A P_D + 2P_A P_U} \end{cases} .$$

Since SA of service S is the time not spent in state F, SA can be given by the formula [Ros2009, Ma2004]:

$$SA = 1 - \frac{X_F H_F}{\mathbf{X}.\mathbf{H}} \qquad (6.8).$$

Plugging the solution for **X** obtained for (6.7) into (6.8) yields the following expression for SA:

$$SA = 1 - \frac{P_A P_U H_F}{H_G + P_A H_A + P_{FP} H_{FP} + P_A P_D H_D + P_A P_U H_U + P_A P_U H_F}$$

$$= \frac{H_G + P_A H_A + P_{FP} H_{FP} + P_A P_D H_D + P_A P_U H_U}{H_G + P_A H_A + P_{FP} H_{FP} + P_A P_D H_D + P_A P_U H_U + P_A P_U H_F} \qquad (6.9).$$

6.4.2 Variation of S-Availability

Let $\Delta_4 = H_G + P_A H_A + P_{FP} H_{FP} + P_A P_D H_D + P_A P_U H_U + P_A P_U H_F$ and compute the partial derivatives of SA with respects to P_A, P_{FP} and P_D.

$$\frac{\partial(SA)}{\partial P_A} = -\frac{1}{(\Delta_4)^2}(P_U H_F \Delta_4 - P_A P_U H_F (H_A + P_D H_D + P_U H_U + P_U H_F))$$

$$\frac{\partial(\mathrm{SA})}{\partial \mathrm{P_A}} = -\frac{\mathrm{P_U H_F}}{(\Delta_4)^2}(\mathrm{H_G} + \mathrm{P_{FP}\, X_{FP}}) < 0 \qquad (6.10).$$

$$\frac{\partial(\mathrm{SA})}{\partial \mathrm{P_{FP}}} = \frac{\mathrm{H_{FP}}}{(\Delta_4)^2} > 0 \qquad (6.11).$$

$$\frac{\partial(\mathrm{SA})}{\partial \mathrm{P_D}} = -\frac{1}{(\Delta_4)^2}(-\mathrm{P_A H_F}\Delta_8 - \mathrm{P_A P_U H_F}(\mathrm{P_A H_D} - \mathrm{P_A H_U} - \mathrm{P_A H_F}))$$

$$\frac{\partial(\mathrm{SA})}{\partial \mathrm{P_D}} = \frac{\mathrm{P_A H_F}}{(\Delta_4)^2}(\Delta_8 + \mathrm{P_A P_U}(\mathrm{H_D} - \mathrm{H_U} - \mathrm{H_F}))$$

$$\frac{\partial(\mathrm{SA})}{\partial \mathrm{P_D}} = \frac{\mathrm{P_A H_F}}{(\Delta_4)^2}(\mathrm{H_G} + \mathrm{P_A H_A} + \mathrm{P_{FP}\, H_{FP}} + \mathrm{P_A P_D H_D} + \mathrm{P_A P_U H_D}) > 0 \quad (6.12).$$

From all these four partial derivatives in (6.10)-(6.12), I can infer that SA of service S will increase if $\mathrm{P_A}$ decreases. In addition, the increase of $\mathrm{P_{FP}}$ and $\mathrm{P_D}$ will have good impact on SA in the sense that it will increase SA.

6.5 Combination of SCIT and IDS

6.5.1 Semi-Markov Model

The transition diagram for the combination of SCIT and IDS (SCIT+IDS) is depicted in Figure 6.2:

- G (Good): Service S starts with a Good state in which it functions normally.
- A (Attacked): As vulnerabilities are exploited, Service S is under attack.
- FP (False Positive): With the IDS in place, a false alarm may be detected, triggering the online service instance to go to cleansing and be replaced by a fresh instance. Hence, the service can be in Good state again.

- D (Detected): With the IDS in place, the attack in state A can be detected, triggering online service instance to go to cleansing and be replaced by a fresh instance. Hence, the service can be in Good state again.
- F (Fail): If the attack is not detected and cleansing does not happen in time, then Service S will fall into Failure mode.

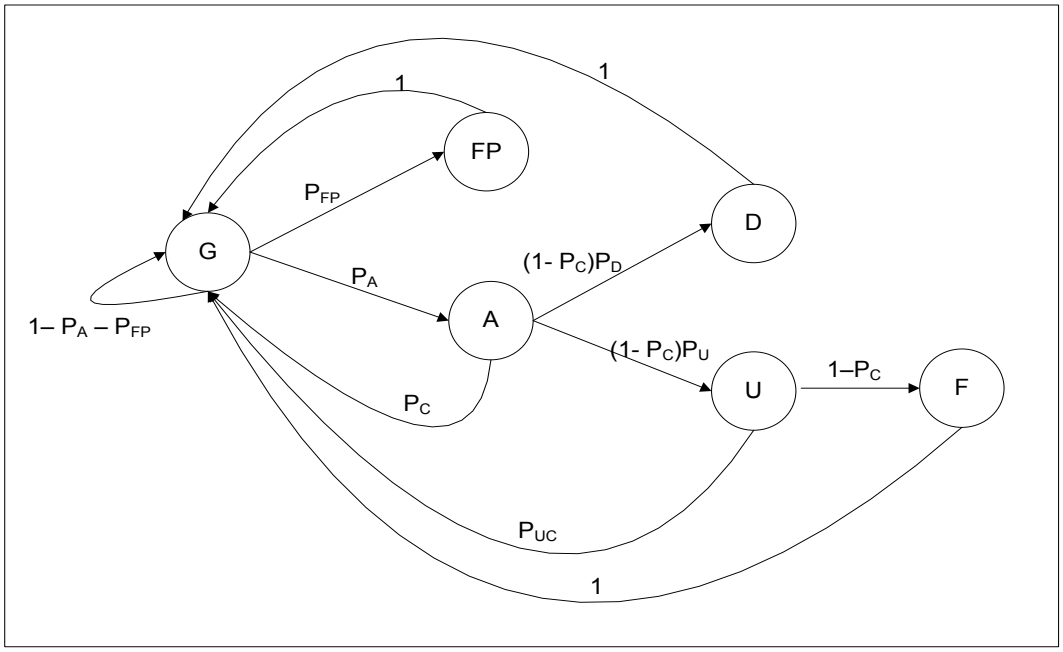

Figure 6.2. Transition Diagram of a Service with SCIT+IDS

Thus, the state space contains the following elements:

$$E = \{G, A, FP, D, U, F\}$$

In terms of transition probabilities, the transition diagram shows:

- P_A as the probability that an attack occurs so that service S goes from state G to state A.
- P_C as the probability of that the online service instance being under attack goes to cleansing mode, and is replaced by a pristine service instance. This makes service S recover to state G from state A.
- P_{FP} as the probability of a false positive, which makes the service go to state FP, then to cleansing mode.
- P_D as the probability of detection. Service S transitions from state A to state D with probability $\overline{P_C}P_D$, where $\overline{P_C} = 1 - P_C$.
- P_U as the probability of detection miss. Service S moves from state A to state U with probability $\overline{P_C}P_U$, where $\overline{P_C} = 1 - P_C$. It is clear that $P_U = 1 - P_D = \overline{P_D}$.
- P_{UC} as the probability of the service being undetected goes to cleansing mode, and is replaced by a pristine service instance. This makes service S recover to state G from state U.
- $\overline{P_{AFP}} = 1 - P_A - P_{FP}$ as the probability that service S at state G stays in G.

For the service S under (SCIT+IDS), the transition probability matrix **P** can be written as:

$$\boldsymbol{P} = \begin{array}{c} \\ G \\ A \\ FP \\ D \\ U \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} G & A & FP & D & U & F \end{array} \\ \begin{pmatrix} \overline{P_{AFP}} & P_A & P_{FP} & 0 & 0 & 0 \\ P_C & 0 & 0 & \overline{P_C}P_D & \overline{P_C}P_U & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ P_{UC} & 0 & 0 & 0 & 0 & \overline{P_{UC}} \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}.$$

6.5.2 MTTSF Analysis

In order to compute the Mean Time to Security Failure (MTTSF) for service S, the state F is considered as an absorbing one.

Within this context, the state pace E is partitioned into the set of transient states E_T and the set of absorbing states E_A:

$$E_T = \{G, A, FP, D, U\} \text{ and}$$

$$E_A = \{F\}.$$

With these two sets E_T and E_A, the transition probability matrix **Q** becomes:

$$\boldsymbol{Q} = \begin{array}{c} \\ G \\ A \\ FP \\ D \\ U \end{array} \begin{array}{c} \begin{array}{ccccc} G & A & FP & D & U \end{array} \\ \begin{pmatrix} \overline{P_{AFP}} & P_A & P_{FP} & 0 & 0 \\ P_C & 0 & 0 & \overline{P_C}P_D & \overline{P_C}P_U \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ P_{UC} & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}.$$

Let:

- **X** = (X_G X_A X_{FP} X_D X_U) be the row vector containing the number of visits at the transient in E_T;
- **H** = (H_G H_A H_{FP} H_D H_U) the row vector containing the mean sojourn times at the transient in E_T;
- $\mathbf{X_0}$ = (1 0 0 0 0) being the starting vector.

Then, the number of visits X can be found by solving the system of equations using the methodology described in [Lim2001, Ma2004]:

$$\mathbf{X} = \mathbf{X_0} + \mathbf{XQ} \qquad (6.13)$$

Equation (6.13) can be expanded as:

$$(X_G\ X_A\ X_{FP}\ X_D\ \ X_U)$$

$$= (1\ \ 0\ \ \ 0\ \ \ 0\ \ 0)$$

$$+\ (X_G\ X_A\ X_{FP}\ X_D\ \ X_U)\begin{pmatrix} \overline{P_{AFP}} & P_A & P_{FP} & 0 & 0 \\ P_C & 0 & 0 & \overline{P_C}P_D & \overline{P_C}P_U \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ P_{UC} & 0 & 0 & 0 & 0 \end{pmatrix},$$

yielding the equations with unknowns X_G, X_A, X_{FP}, X_D and X_U:

$$(6.13) \Leftrightarrow \begin{cases} X_G = 1 + X_G\overline{P_{AFP}} + X_A P_C + X_{FP} + X_D + X_U P_{UC} \\ X_A = X_G P_A \\ X_{FP} = X_G P_{FP} \\ X_D = X_A\overline{P_C}P_D \\ X_U = X_A\overline{P_C}P_U \end{cases}$$

$$\Leftrightarrow \begin{cases} X_G = 1 + X_G\overline{P_{AFP}} + X_A P_C + X_{FP} + X_D + X_U P_{UC} \\ X_A = X_G P_A \\ X_{FP} = X_G P_{FP} \\ X_D = X_G P_A\overline{P_C}P_D \\ X_U = X_G P_A\overline{P_C}P_U \end{cases}$$

$$\Leftrightarrow \begin{cases} X_G = 1 + X_G\overline{P_{AFP}} + X_G P_A P_C + X_G P_{FP} + X_G P_A\overline{P_C}P_D + X_G P_A\overline{P_C}P_U P_{UC} \\ X_A = X_G P_A \\ X_{FP} = X_G P_{FP} \\ X_D = X_G P_A\overline{P_C}P_D \\ X_U = X_G P_A\overline{P_C}P_U \end{cases}$$

$$\Leftrightarrow \begin{cases} X_G = \dfrac{1}{\Delta_6} \\ X_A = X_G P_A \\ X_{FP} = X_G P_{FP} \\ X_D = X_G P_A\overline{P_C}P_D \\ X_U = X_G P_A\overline{P_C}P_U \end{cases},$$

where $\Delta_6 = 1 - \overline{P_{AFP}} - P_A P_C - P_{FP} - P_A \overline{P_C} P_D - P_A \overline{P_C} P_U P_{UC}$. Δ_6 can be simplified further:

$$\Delta_6 = 1 - (1 - P_A - P_{FP}) - P_A P_C - P_{FP} - P_A \overline{P_C} P_D - P_A \overline{P_C} P_U P_{UC}$$

$$\Delta_6 = P_A - P_A P_C - P_A \overline{P_C} P_D - P_A \overline{P_C} P_U P_{UC}$$

$$\Delta_6 = P_A \overline{P_C}(1 - P_D - P_U P_{UC})$$

$$\Delta_6 = P_A \overline{P_C}(P_U - P_U P_{UC})$$

$$\Delta_6 = P_A \, \overline{P_C} P_U \overline{P_{UC}}$$

According to [Ros2009, Tri2002, Ma2004], MTTSF is given by the scalar product:

$$\text{MTTSF} = \mathbf{X} \,.\, \mathbf{H}$$

Then, plugging the solution for **X** in (6.13) gives us the following expression for MTTSF:

$$\text{MTTSF} = X_G H_G + X_A H_A + X_{FP} H_{FP} + X_D H_D + X_U H_U$$

$$= \frac{H_G + P_A H_A + P_{FP} H_{FP} + P_A \overline{P_C} P_D H_D + P_A \overline{P_C} P_U H_U}{\Delta_6} \qquad (6.14).$$

6.5.3 Variation of MTTSF

According to my study in the case of an atomic service protected by simple SCIT, MTTSF increases when P_A decreases and P_C increases. Let's check that it is still the case for the atomic service protected by (SCIT+IDS). Computing the partial derivative of MTTSF in (6.14) with respect to P_A:

$$\frac{\partial(\text{MTTSF})}{\partial P_A} = \frac{1}{{\Delta_6}^2}\{(H_A + \overline{P_C} P_D H_D + \overline{P_C} P_U H_U) P_A \, \overline{P_C} P_U \overline{P_{UC}}$$

$$- \overline{P_C} P_U \overline{P_{UC}}(H_G + P_A H_A + P_{FP} H_{FP} + P_A \overline{P_C} P_D H_D + P_A \overline{P_C} P_U H_U)\}$$

$$\frac{\partial(\mathrm{MTTSF})}{\partial \mathrm{P_A}} = \frac{\bar{\mathrm{P_C}}\mathrm{P_U}\overline{\mathrm{P_{UC}}}}{\Delta_6{}^2}\{(\mathrm{H_A} + \bar{\mathrm{P_C}}\mathrm{P_D}\mathrm{H_D} + \bar{\mathrm{P_C}}\mathrm{P_U}\mathrm{H_U})\mathrm{P_A}$$

$$- (\mathrm{H_G} + \mathrm{P_A}\mathrm{H_A} + \mathrm{P_{FP}}\mathrm{H_{FP}} + \mathrm{P_A}\bar{\mathrm{P_C}}\mathrm{P_D}\mathrm{H_D} + \mathrm{P_A}\bar{\mathrm{P_C}}\mathrm{P_U}\mathrm{H_U})\}$$

$$\frac{\partial(\mathrm{MTTSF})}{\partial \mathrm{P_A}} = -\frac{\bar{\mathrm{P_C}}\mathrm{P_U}\overline{\mathrm{P_{UC}}}}{\Delta_6{}^2}(\mathrm{H_G} + \mathrm{P_{FP}}\mathrm{H_{FP}}) < 0 \qquad (6.15).$$

Let's compute the partial derivative of MTTSF with respect to $\mathrm{P_C}$.

$$\frac{\partial(\mathrm{MTTSF})}{\partial \mathrm{P_C}} = \frac{1}{\Delta_6{}^2}\{(-\mathrm{P_A}\,\mathrm{P_D}\mathrm{H_D} - \mathrm{P_A}\,\mathrm{P_U}\mathrm{H_U})\mathrm{P_A}\,\bar{\mathrm{P_C}}\mathrm{P_U}\overline{\mathrm{P_{UC}}}$$

$$+ (\mathrm{H_G} + \mathrm{P_A}\mathrm{H_A} + \mathrm{P_{FP}}\mathrm{H_{FP}} + \mathrm{P_A}\bar{\mathrm{P_C}}\mathrm{P_D}\mathrm{H_D} + \mathrm{P_A}\bar{\mathrm{P_C}}\mathrm{P_U}\mathrm{H_U})\mathrm{P_A}\,\mathrm{P_U}\overline{\mathrm{P_{UC}}}\}$$

$$\frac{\partial(\mathrm{MTTSF})}{\partial \mathrm{P_C}} = \frac{1}{\Delta_6{}^2}(\mathrm{H_G} + \mathrm{P_A}\mathrm{H_A} + \mathrm{P_{FP}}\mathrm{H_{FP}})(\mathrm{P_A}\,\overline{\mathrm{P_{UC}}}\mathrm{P_U}) > 0 \quad (6.16).$$

Computing the partial derivative of MTTSF with respect to $\mathrm{P_{UC}}$ will yield a result similar to (6.16).

The study of the impact of the control parameters of IDS can also be done by computing the partial derivatives of MTTSF with respect to $\mathrm{P_{FP}}$ and $\mathrm{P_D}$.

$$\frac{\partial(\mathrm{MTTSF})}{\partial \mathrm{P_{FP}}} = \frac{1}{\Delta_6}(\mathrm{H_{FP}}) > 0 \qquad (6.17).$$

$$\frac{\partial(\mathrm{MTTSF})}{\partial \mathrm{P_D}} = \frac{1}{\Delta_6{}^2}\{(\mathrm{P_A}\,\mathrm{P_C}\mathrm{H_D} - \mathrm{P_A}\bar{\mathrm{P_C}}\mathrm{H_U})\mathrm{P_A}\,\bar{\mathrm{P_C}}\mathrm{P_U}\overline{\mathrm{P_{UC}}}$$

$$+ (\mathrm{H_G} + \mathrm{P_A}\mathrm{H_A} + \mathrm{P_{FP}}\mathrm{H_{FP}} + \mathrm{P_A}\bar{\mathrm{P_C}}\mathrm{P_D}\mathrm{H_D} + \mathrm{P_A}\bar{\mathrm{P_C}}\mathrm{P_U}\mathrm{H_U})\mathrm{P_A}\,\bar{\mathrm{P_C}}\overline{\mathrm{P_{UC}}}\}$$

$$\frac{\partial(\mathrm{MTTSF})}{\partial \mathrm{P_D}} = \frac{\mathrm{P_A}\,\bar{\mathrm{P_C}}\overline{\mathrm{P_{UC}}}}{\Delta_6{}^2}\{(\mathrm{P_A}\,\mathrm{P_C}\mathrm{H_D})\mathrm{P_U} + (\mathrm{H_G} + \mathrm{P_A}\mathrm{H_A} + \mathrm{P_{FP}}\mathrm{H_{FP}} + \mathrm{P_A}\bar{\mathrm{P_C}}\mathrm{P_D}\mathrm{H_D})\}$$

$$> 0 \quad (6.18).$$

From all these four partial derivatives in (6.15)-(6.18), I can infer that:

- MTTSF of service S can be controlled by the attack probability P_A on S and its cleaning probability P_C. This result has been obtained in the case of an atomic service, so it's not new.
- P_{FP} and P_D don't have adverse effects on MTTSF. More precisely, MTTSF will increase if these probabilities increase.

6.5.4 S-Availabilitty Analysis

The derivation of the expression for the S-Availability (SA) can be done using the same approach as in the atomic service. By not considering state F as an absorbing state, I will use the entire transition matrix **P** encompassing all states in E = {G, A, FP, D, U, F}.

Let:

- **X** = (X_G X_A X_{FP} X_D X_U X_F) be the row vector containing the steady-state probabilities of states G, A and F in E respectively in the embedded Markov chain;
- **H** = (H_G H_A H_{FP} H_D H_U H_{FF}) the row vector containing the mean sojourn times at the states in E.

The steady-state probabilities can be found by solving the system of equations:

$$\begin{cases} \mathbf{X} = \mathbf{XP} \\ \sum_{i \in E} X_i = 1 \end{cases} \qquad (6.19)\cdot$$

The matrix equation X = XP in (6.19) can be expanded as:

$$\{(X_G\ X_A\ X_{FP}\ X_D\ X_U\ X_F) = (X_G\ X_A\ X_{FP}\ X_D\ X_U\ X_F)\begin{pmatrix} \overline{P_{AFP}} & P_A & P_{FP} & 0 & 0 & 0 \\ P_C & 0 & 0 & \overline{P_C}P_D & \overline{P_C}P_U & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ P_{UC} & 0 & 0 & 0 & 0 & \overline{P_{UC}} \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

$$(6.19) \Leftrightarrow \begin{cases} X_G = X_G\overline{P_{AFP}} + X_AP_C + X_{FP} + X_D + X_UP_{UC} + X_F \\ X_A = X_GP_A \\ X_{FP} = X_GP_{FP} \\ X_D = X_A\overline{P_C}P_D \\ X_U = X_A\overline{P_C}P_U \\ X_F = X_U\overline{P_{UC}} \\ X_G + X_A + X_{FP} + X_D + X_U + X_F = 1 \end{cases}$$

$$\Leftrightarrow \begin{cases} X_G = X_G\overline{P_{AFP}} + X_AP_C + X_{FP} + X_D + X_UP_{UC} + X_F \\ X_A = X_GP_A \\ X_{FP} = X_GP_{FP} \\ X_D = X_GP_A\overline{P_C}P_D \\ X_U = X_GP_A\overline{P_C}P_U \\ X_F = X_GP_A\overline{P_C}P_U\overline{P_{UC}} \\ X_G + X_A + X_{FP} + X_D + X_U + X_F = 1 \end{cases}$$

$$\Leftrightarrow \begin{cases} X_A = X_GP_A \\ X_{FP} = X_GP_{FP} \\ X_D = X_GP_A\overline{P_C}P_D \\ X_U = X_GP_A\overline{P_C}P_U \\ X_F = X_GP_A\overline{P_C}P_U\overline{P_{UC}} \\ X_G = \dfrac{1}{1 + P_A + P_{FP} + P_A\overline{P_C}P_D + P_A\overline{P_C}P_U + P_A\overline{P_C}P_U\overline{P_{UC}}} \end{cases}$$

I consider SA of service S within the composite service is the time not spent in state F. Thus, SA can be given by the formula [Ros2009, Ma2004]:

$$SA = 1 - \frac{X_FH_F}{\mathbf{X.H}} \qquad (6.20).$$

Plugging the solution for **X** expressed in (6.19) into (6.20) yields the following expression for SA:

$$SA = 1 - \frac{P_A \overline{P_C} P_U \overline{P_{UC}} H_F}{\Delta_7} \qquad (6.21),$$

where $\Delta_7 = H_G + P_A H_A + P_{FP} H_{FP} + P_A \overline{P_C} P_D H_D + P_A \overline{P_C} P_U H_U + P_A \overline{P_C} P_U \overline{P_{UC}} H_F$.

6.5.5 Variation of S-Availability

Computing the partial derivative of SA in (6.21) with respect to P_A:

$$\frac{\partial(SA)}{\partial P_A} = -\frac{1}{{\Delta_7}^2}\{\overline{P_C} P_U \overline{P_{UC}} H_F \Delta_7 - P_A \overline{P_C} P_U \overline{P_{UC}} H_F (H_A + \overline{P_C} P_D H_D + \overline{P_C} P_U H_U + \overline{P_C} P_U \overline{P_{UC}} H_F)\}$$

$$\frac{\partial(SA)}{\partial P_A} = -\frac{\overline{P_C} P_U \overline{P_{UC}} H_F}{{\Delta_7}^2}(H_G + P_{FP} H_{FP}) < 0 \qquad (6.22).$$

Let's compute the partial derivative of SA with respect to P_C.

$$\frac{\partial(SA)}{\partial P_C} = -\frac{1}{{\Delta_7}^2}\{-P_A \overline{P_{UC}} P_U H_F \Delta_7 - P_A \overline{P_C}\ \overline{P_{UC}} P_U H_F (-P_A P_D H_D - P_A P_U H_U - P_A P_U \overline{P_{UC}} H_F)\}$$

$$\frac{\partial(SA)}{\partial P_C} = \frac{P_A \overline{P_{UC}} P_U H_F}{{\Delta_7}^2}(\Delta_7 - \overline{P_C}(P_A P_D H_D + P_A P_U H_U + P_A P_U \overline{P_{UC}} H_F))$$

$$\frac{\partial(SA)}{\partial P_C} = \frac{P_A \overline{P_{UC}} P_U H_F}{{\Delta_7}^2}(H_G + P_A H_A + P_{FP} H_{FP} + P_A \overline{P_C} P_D H_D + P_A \overline{P_C} P_U H_U + P_A \overline{P_C} P_U \overline{P_{UC}} H_F$$

$$- \overline{P_C}(P_A P_D H_D + P_A P_U H_U + P_A P_U \overline{P_{UC}} H_F))$$

$$\frac{\partial(SA)}{\partial P_C} = \frac{P_A \overline{P_{UC}} P_U H_F}{{\Delta_7}^2}(H_G + P_A H_A + P_{FP} H_{FP}) > 0 \quad (6.23).$$

The sign of the partial derivative of SA with respect to P_{UC} can be proven to be positive, using similar computations.

The study of the impact of the control parameters of IDS can also be done by computing the partial derivatives of SA with respect to P_{FP} and P_D.

$$\frac{\partial(SA)}{\partial P_{FP}} = \frac{P_A \bar{P_C} P_U \overline{P_{UC}} H_{FP}}{{\Delta_7}^2} > 0 \qquad (6.24).$$

$$\frac{\partial(SA)}{\partial P_D} = -\frac{1}{{\Delta_7}^2}\{-P_A \bar{P_C}\overline{P_{UC}} H_F \Delta_7 - (P_A \bar{P_C} H_D - P_A \bar{P_C} H_U - P_A \bar{P_C}\overline{P_{UC}} H_F) P_A \bar{P_C} P_U \overline{P_{UC}} H_F\}$$

$$\frac{\partial(SA)}{\partial P_D} = \frac{P_A \bar{P_C}\overline{P_{UC}} H_F}{{\Delta_7}^2}\{H_G + P_A H_A + P_{FP} H_{FP} + P_A \bar{P_C} P_D H_D + P_A \bar{P_C} P_U H_U + P_A \bar{P_C}\overline{P_{UC}} P_U H_F$$
$$+ (P_A \bar{P_C} H_D - P_A \bar{P_C} H_U - P_A \bar{P_C}\overline{P_{UC}} H_F) P_U\}$$

$$\frac{\partial(SA)}{\partial P_D} = \frac{P_A {\bar{P_C}}^2 H_F}{{\Delta_7}^2}\{H_G + P_A H_A + P_{FP} H_{FP} + P_A \bar{P_C} P_D H_D + P_A \bar{P_C} P_U H_D\} < 0 \quad (6.25).$$

From all these four partial derivatives in (6.22)-(6.25), I can infer that:

- SA of service S can be controlled by the attack probability P_A on S and its cleaning probability P_C. This result has been obtained in the case of an atomic service, so it's not new.
- P_{FP} and P_D don't have adverse effects on SA; that is, SA will increase if these probabilities increase.

6.5.6 Other IT-QoS

For other IT-QoS, the study is the same as in the case of an atomic service. So, I will not repeat it here.

6.6 Compare IDS and (SCIT+IDS)

I have obtained the IT-QoS expressions for MTTSF and SA when using IDS alone and the combination (SCIT+IDS). Let's study how the latter combination help the system

designer to have better control of these two IT-QoS parameters. The benefits of the combination (SCIT+IDS) have been shown in [Nag2010] and [Nag2011]: the former utilizes the decision tree and associated probabilities, and the latter proves the advantages of the combined strategy by analyzing the costs of IDS and (SCIT+IDS).

6.6.1 MTTSF Comparison

Let's denote $MTTSF_1$ and $MTTSF_2$ be the MTTSF in the two cases of IDS and (SCIT+IDS) respectively. From expressions (6.3) and (6.14), I can write the difference between $MTTSF_2$ and $MTTSF_1$ as:

$$\begin{aligned} &\mathrm{MTTSF_2} - \mathrm{MTTSF_1} \\ &= \frac{\mathrm{H_G} + \mathrm{P_A H_A} + \mathrm{P_{FP} H_{FP}} + \mathrm{P_A}\overline{\mathrm{P_C}}\mathrm{P_D H_D} + \mathrm{P_A}\overline{\mathrm{P_C}}\mathrm{P_U H_U}}{\mathrm{P_A P_U}\overline{\mathrm{P_C}}\,\overline{\mathrm{P_{UC}}}} \\ &\quad - \frac{\mathrm{H_G} + \mathrm{P_A H_A} + \mathrm{P_{FP} H_{FP}} + \mathrm{P_A P_D H_D} + \mathrm{P_A P_U H_U}}{\mathrm{P_A P_U}} \\ &= \frac{(\mathrm{H_G} + \mathrm{P_A H_A} + \mathrm{P_{FP} H_{FP}})(1 - \overline{\mathrm{P_C}}\,\overline{\mathrm{P_{UC}}}) + \mathrm{P_A}\overline{\mathrm{P_C}}(1 - \overline{\mathrm{P_{UC}}})(\mathrm{P_D H_D} + \mathrm{P_U H_U})}{\mathrm{P_A P_U}\overline{\mathrm{P_C}}\,\overline{\mathrm{P_{UC}}}} \\ &> 0 \quad (6.26). \end{aligned}$$

The inequality (6.26) shows that for the same operational values of P_{FP} and P_D of IDS, MTTSF becomes greater if SCIT is used in conjunction with IDS.

The next question is how much increase can be obtained by adding SCIT to IDS as security protection mechanism. For this, let's compute the partial derivative of $|MTTSF_2 - MTTSF_1| = (MTTSF_2 - MTTSF_1)$ with respect to P_A and P_C. First:

$$\frac{\partial(\mathrm{MTTSF_2} - \mathrm{MTTSF_1})}{\partial \mathrm{P_C}} = \frac{\partial(\mathrm{MTTSF_2})}{\partial \mathrm{P_C}} - \frac{\partial(\mathrm{MTTSF_1})}{\partial \mathrm{P_C}} = \frac{\partial(\mathrm{MTTSF_2})}{\partial \mathrm{P_C}} > 0 \qquad (6.27),$$

according to (6.16).

Second:

$$\frac{\partial(\mathrm{MTTSF_2} - \mathrm{MTTSF_1})}{\partial \mathrm{P_A}} = \frac{\partial(\mathrm{MTTSF_2})}{\partial \mathrm{P_A}} - \frac{\partial(\mathrm{MTTSF_1})}{\partial \mathrm{P_A}}$$

$$= -\frac{\bar{\mathrm{P}}_\mathrm{C}\mathrm{P_U}\overline{\mathrm{P_{UC}}}}{(\mathrm{P_A}\,\bar{\mathrm{P}}_\mathrm{C}\mathrm{P_U}\overline{\mathrm{P_{UC}}})^2}(\mathrm{H_G} + \mathrm{P_{FP}H_{FP}}) - \frac{-(\mathrm{H_G} + \mathrm{P_{FP}H_{FP}})\mathrm{P_U}}{(\mathrm{P_A P_U})^2}$$

$$= -\frac{\mathrm{H_G} + \mathrm{P_{FP}H_{FP}}}{\mathrm{P_A}^2\bar{\mathrm{P}}_\mathrm{C}\mathrm{P_U}\bar{\mathrm{P}}_\mathrm{UC}}\,(1 - \bar{\mathrm{P}}_\mathrm{C}\bar{\mathrm{P}}_\mathrm{UC}) < 0 \qquad (6.28).$$

From (6.27) and (6.28), I can deduce that the difference $|\mathrm{MTTSF_2} - \mathrm{MTTSF_1}|$ increases as $\mathrm{P_C}$ increases and $\mathrm{P_A}$ decreases. With SCIT, the probabilities $\mathrm{P_A}$ and $\mathrm{P_C}$ can be controlled via SCIT's exposure window $\mathrm{W_O}$. That is, decreasing the exposure window will make the difference $|\mathrm{MTTSF_2} - \mathrm{MTTSF_1}|$ larger. Thus, there are advantages in using SCIT in combination of IDS.

6.6.2 S-Availability Comparison

Let's denote $\mathrm{SA_1}$ as the SA in the case of IDS, and $\mathrm{SA_2}$ the counterpart expression in the case of (SCIT+IDS). From expressions (6.9) and (6.21), I can write the difference between $\mathrm{SA_2}$ and $\mathrm{SA_1}$ as:

$$\mathrm{SA_2} - \mathrm{SA_1} = -\frac{\mathrm{P_A}\bar{\mathrm{P}}_\mathrm{C}\mathrm{P_U}\bar{\mathrm{P}}_\mathrm{UC}\mathrm{H_F}}{\Delta_7} + \frac{\mathrm{P_A P_U H_F}}{\Delta_4},$$

where $\Delta_7 = \mathrm{H_G} + \mathrm{P_A H_A} + \mathrm{P_{FP}H_{FP}} + \mathrm{P_A}\bar{\mathrm{P}}_\mathrm{C}\mathrm{P_D H_D} + \mathrm{P_A}\bar{\mathrm{P}}_\mathrm{C}\mathrm{P_U H_U} + \mathrm{P_A}\bar{\mathrm{P}}_\mathrm{C}\mathrm{P_U}\bar{\mathrm{P}}_\mathrm{UC}\mathrm{H_F}$ and

$\Delta_4 = \mathrm{H_G} + \mathrm{P_A H_A} + \mathrm{P_{FP}\,H_{FP}} + \mathrm{P_A P_D H_D} + \mathrm{P_A P_U H_U} + \mathrm{P_A P_U H_F}$.

$$\mathrm{SA_2} - \mathrm{SA_1} = -\frac{\mathrm{P_A P_U H_F}}{\Delta_7\Delta_4}\,(\bar{\mathrm{P}}_\mathrm{C}\bar{\mathrm{P}}_\mathrm{UC}\Delta_4 - \Delta_7)$$

$$= -\frac{P_A P_U H_F}{\Delta_7 \Delta_8} \{(\bar{P_C}\bar{P}_{UC} - 1)(H_G + P_A H_A + P_{FP} H_{FP}) - P_A \bar{P_C} P_{UC}(P_D H_D + P_U H_U)\}$$

$$> 0 \quad (6.29).$$

The inequality (6.29) shows that for the same operational values of P_{FP} and P_D of IDS, SA becomes larger if SCIT is used in conjunction with IDS.

The next question is how much increase in SA can be obtained by adding SCIT to IDS as the combined security protection mechanism. Since $SA_2 > SA_1$ according to (6.29), let's compute the partial derivative of (SA_2 - SA_1) with respect to P_C and P_A. First:

$$\frac{\partial(SA_2 - SA_1)}{\partial P_C} = \frac{\partial(SA_2)}{\partial P_C} - \frac{\partial(SA_1)}{\partial P_C} = \frac{\partial(SA_2)}{\partial P_C} > 0 \quad (6.30),$$

according to (6.23).

Secondly, I can write:

$$\frac{\partial(SA_2 - SA_1)}{\partial P_A} = \frac{\partial(SA_2)}{\partial P_A} - \frac{\partial(SA_1)}{\partial P_A}$$

$$= \frac{\bar{P_C} P_U \bar{P}_{UC} H_F}{{\Delta_7}^2}(H_G + P_{FP} H_{FP}) - \frac{P_U H_F}{{\Delta_4}^2}(H_G + P_{FP} X_{FP})$$

$$= P_U H_F (H_G + P_{FP} H_{FP}) \left(\frac{\bar{P_C}\bar{P}_{UC}}{{\Delta_7}^2} - \frac{1}{{\Delta_4}^2}\right)$$

$$= P_U H_F (H_G + P_{FP} H_{FP}) \left(\frac{1}{\Delta_4} + \frac{\sqrt{\bar{P_C}\bar{P}_{UC}}}{\Delta_7}\right)\left(-\frac{1}{\Delta_4} + \frac{\sqrt{\bar{P_C}\bar{P}_{UC}}}{\Delta_7}\right)$$

$$= \frac{P_U H_F (H_G + P_{FP} H_{FP})}{\Delta_7 \Delta_4} \left(\frac{1}{\Delta_4} + \frac{\sqrt{\bar{P_C}\bar{P}_{UC}}}{\Delta_7}\right)(-\Delta_7 + \Delta_4 \sqrt{\bar{P_C}\bar{P}_{UC}})$$

Thus,

$$\frac{\partial(\mathrm{SA}_2 - \mathrm{SA}_1)}{\partial \mathrm{P_A}} > 0 \Leftrightarrow -\Delta_7 + \Delta_4\sqrt{\overline{\mathrm{P_C}}\,\overline{\mathrm{P}}_{\mathrm{UC}}} > 0$$

$$\Leftrightarrow \frac{\Delta_7}{\Delta_4\sqrt{\overline{\mathrm{P_C}}\,\overline{\mathrm{P}}_{\mathrm{UC}}}} < 1 \quad (6.31).$$

From (6.30), I can deduce that the difference $|SA_1 - SA_2|$ increases as P_C increases and P_A decreases. With respect to P_A, the gap $(SA_2 - SA_1)$ will be increased if P_A decreases and relation (6.31) is satisfied. With SCIT, the probabilities P_A and P_C can be controlled via SCIT's exposure window W. That is, decreasing the exposure window will make S-Availability SA_2 much larger than the S-Availability SA_1 of using IDS alone. Thus, for SA, I can reach the same conclusion arrived when studying MTTSF above.

6.7 Numerical Examples

In the following numerical examples, I will use the Poisson model only. In Chapter 4, the expressions for P_A, P_C, H_G, H_A, and H_F have been obtained. The expression for P_{UC} can be derived by adapting Theorem 4.4:

$$P_{UC} = \frac{1}{\lambda_U W}\left(1 - e^{-\lambda_U W}\right),$$

where λ_U is the parameter of the exponential distribution for the time an attack remains undetected. Similarly, the methods of Proposition 4.5 can yield the expressions for H_U, H_{FP}, and H_D:

$$\begin{cases} H_U = \dfrac{1}{\lambda_U} - \dfrac{1}{{\lambda_U}^2 W}(1 - e^{-\lambda_U W}) \\ H_{FP} = \dfrac{W}{2} \\ H_D = \dfrac{W}{2} \end{cases}.$$

For example, let $\lambda_A = \lambda_F = 10^{-1}$ and $\lambda_U = 10^{-1}$, $P_D = 0.90$ and $P_{FP} = 0.1$. From Figure 6.3, we notice that with the given parameters regarding the attack arrival, detection probability and false positive probability, the combination (SCIT+IDS) enhances MTTSF by orders of magnitude as compared to the use of either SCIT and IDS alone.

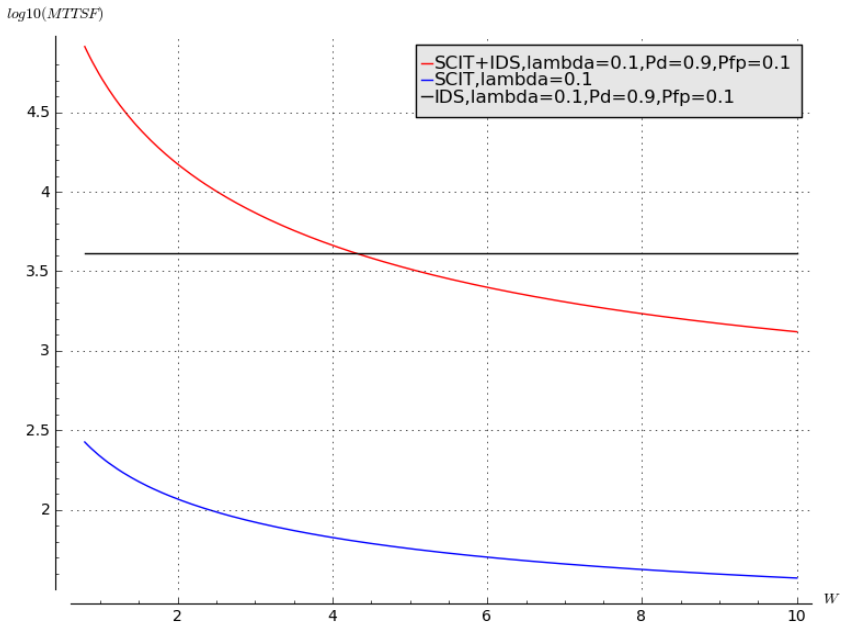

Figure 6.3. Comparison of IDS, SCIT and SCIT+IDS.

The following Figure 6.4 shows the series of MTTSF curves with respect to the exposure window, the false positive probability being fixed, and the detection probability varying. The values for the rates are still the same as above. As expected, better detection leads to a more intrusion tolerant service.

With a fixed value for the detection probability, and various false positive probabilities, Figure 6.5 reveals that the higher the false positive probability is, the better intrusion

tolerant is the service. This can be explained by the fact that an alert, whether true or false, will trigger a cleansing event, which brings the service to a pristine state.

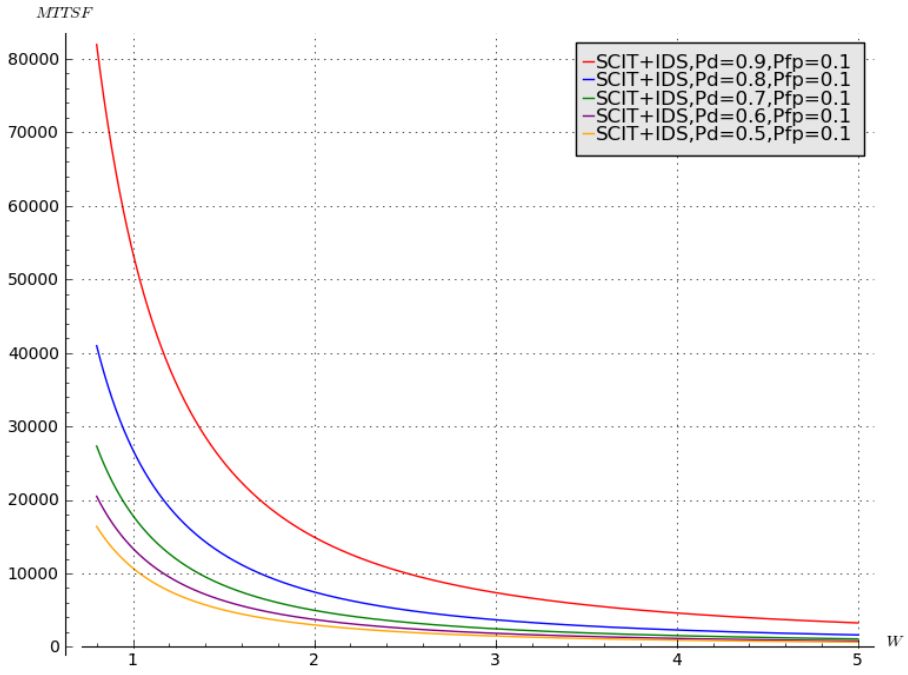

Figure 6.4. SCIT+IDS - Fixed P_{FP} and varying P_D.

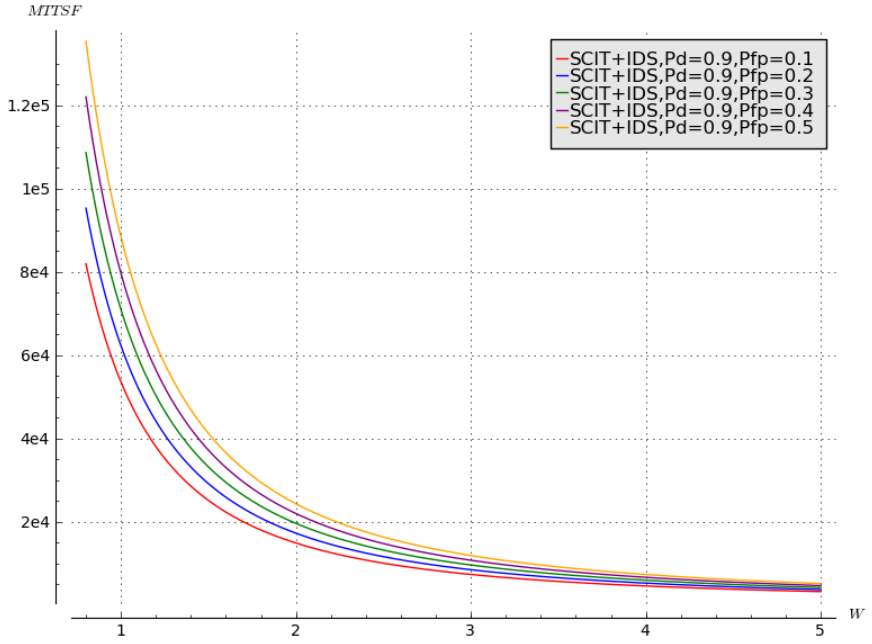

Figure 6.5. SCIT+IDS - Fixed P_D and varying P_{FP}.

6.8 Summary

From the curves above, it appears that (SCIT+IDS) boosts the resilience of the service. I have shown the feasibility of the methodology as applied to an ITS system having the combination of two strategies, namely SCIT and IDS. Moreover, I believe that the analysis of the combination of ITS mechanisms could be generalized to other ITS strategies. From the analysis presented in this chapter, the system designer can do the following to protect service S using the combination (SCIT+IDS):

- select the operating values for P_{FP} and P_D using a ROC curve or some other optimizing method [Axe2000, Nag2011];
- adjust P_C to achieve the desired value of IT-QoS.

CHAPTER 7 - SPACE AND TIME STRATEGY

7.1 Overview

In previous chapters, I have presented Semi-Markov Processes for the analysis of the intrusion tolerance characteristics of a service S, whether atomic or composite, protected by SCIT or the combination (SCIT+IDS). Since SCIT main control parameter is the exposure window, I would call the intrusion tolerance strategy of SCIT is temporal. According to [Man2011, Liu2008], a service is associated to its attack surface, which in turn is determined by its vulnerabilities due to software construction and configuration. As a service is enhanced to add more features, the attack surface may be widened. Within this context of attack surface, I would like to introduce the concept of intrusion tolerance based on space and time dimensions. The time dimension will be provided by SCIT, and will reduce the exposure of the service's attack surface, which can be considered as representing space dimensions. In other words, the approach consists of compensating the undesired expansion of a service's attack surface due to enhancements by manipulating the exposure time. On one hand, it might be impractical to rely solely on mechanisms to minimize attack surface as proposed in [How2002, Liu2008], due to the fact that a service implemented by a COTS product will most likely have its attack surface established. Although it might be possible to have recourse to COTS configuration in order to reduce the effective attack surface at service deployment time,

such approaches have limitations from the intrusion tolerance perspectives. On the other hand, considering both the space and time factors has the potential of providing a synergy to ensure the IT-QoS of a service.

Starting from the stochastic analysis in previous chapters, I will show how the attack surface of a service S can be reduced when S is protected by SCIT or SCIT combined with another intrusion tolerance mechanism. The corollary of the analysis will provide another perspective of the approach to tune a service so that its attackability can be controlled, and the IT-QoS guaranteed despite the exposed attack surface.

7.2 Background

7.2.1 Concept

Intuitively, the *attack surface* of a service represents its exposure to malicious attacks. The exposure includes vulnerabilities and even system components that may be susceptible to exploitations. On the theory side, Manadhata modeled the attack surface aspect of a service by an I/O automaton comprising action signatures, states, transitions, and start state [Man2011]. The I/O automaton fits naturally to the data flow of an attack which, at the abstract level, consists of inputting malicious data in the form of malware and/or outputting data (privacy data, security privileges, system and network configuration elements). Adapting the service to an I/O automaton:

- service operations with their input and output parameters comprise the action signatures;
- states represent the values associated with the variables in the service;

- a transition corresponds to the invocation of an operation, causing a re-association of the values to the variables;
- and start state can be the set of initial values.

With this adaptation, the entry and exit points will make up the attack surface of a service. In [Man2011], the three categories of entry/exit points are methods, channels, and data items, which correspond to service operations, service protocols, and service data respectively. In the literature, these attack points exposed by a service are called *resources* and *dimensions* [Liu2008, How2002, Heu2010].

Another perspective was presented in [Liu2008], where attack surface is viewed more as a system attribute exposed to external clients. By proving the theory that there exists a correlation between the attack surface and internal software qualities such as complexity, service coupling, etc, . Liu arrived at the strategy to reduce the attack surface of a service since the early stage of software development. In other words, software engineers can improve internal software qualities, such as reducing service complexity and coupling between services in order to minimize a service's attack surface.

7.2.2 Metric

For the scope of my research, a metric for a service's attack surface is needed to plug in the quantitative QFITS framework. This measurement can be inserted into the expressions for the IT-QoS studied in previous chapters in various cases of atomic services, composite services, and combination of SCIT and IDS as security protection mechanism. As an attack surface is determined by resources or dimensions, its measurement is a function of the measurements of these dimensions, which can include

methods, processes, channels, protocols, data items, web pages, and access rights. Each software product, system or service may have its own set of dimensions. There are two proposals to quantify an attack surface.

The first one is to express an attack surface as a vector whose entries correspond to various dimensions being considered for the attack surface [Heu2010]. For example, one can have the attack surface metric A expressed by the vector:

$$A = (O, P, D),$$

where O, P, and D are the measurements for operations, protocols, and data items used by the service. Each of these measurements is in turn obtained by summing up the weighted counts. For instance, assume that a service S has two operations GET(data) and PUT(data); these two operations can be executed via either HTTP or FTP protocols; data items are files, some are static pages, and others configuration files.

Table 7.1. Attack Surface - Simple Metric Example

	Name	Weight	Count	Measurement	Total
Operations	GET	1	1	1	4
	PUT	3	1	3	
Protocols	HTTP	1	1	1	3
	FTP	2	1	2	
Data Items	Static Page	1	4	4	15
	Configuration File	4	3	12	

With the weights of the dimensions as given in the above Table 7.1, the attack surface metric of the service is the scalar vector:

$$\mathrm{A} = (4, 3, 15).$$

Instead of using the count, another proposal for the metric is based on the ratio of damage cost over the effort cost spent to compromise a resource [Liu2008, Man2011]. If the same attack effort can compromise a high value resource, then the ratio will be higher than in the case of a lower value resource. The same technique may allow a malicious actor to get hold of the file system on the server. In this scenario, acquiring a configuration file or the password file is obviously more valuable than a HTML file of a static web page, making the dame-effort ratio higher. But, compromising the PUT operation of the service may require much effort than the GET operation due to multiple safeguards; hence, the damage-effort ratio may be equal for both PUT and GET operations. The following table illustrates the example of the metric.

Table 7.2. Attack Surface - Damage-Effort Ratio Example.

	Name	Weight	Damage-Effort Ratio	Measurement
Operations	GET	1	2	8
	PUT	3	2	
Protocols	HTTP	1	1	3
	FTP	2	1	
Data Items	Static Page	1	2	17
	Configuration File	3	5	

In the example contained in Table 7.2, the attack surface metric of the service is the scalar vector:

$$A = (8, 3, 17).$$

The weight assigned to each resource provides to the designer a methodology utilized in risk management to account for the relative impact and criticality in terms of security.

In the Semi-Markov based analysis discussed in previous chapters, I study the behavior of a service in the face of malicious attacks at the service level, without going down to the service's resources. Due to this approach, in order to plug into the analytical expressions obtained for the IT-QoS, I will use a norm function applied on that vector instead of using the measurement vector itself. Such norm function can be the total sum of all the vector entries [How2002] or the Euclidean norm [Heu2010]. Thus, for A = (8, 3, 17), the norm of A, denoted by $\sigma = |A|$ can be calculated as:

- $\sigma = 8 + 3 + 17 = 28$, if the sum norm is used, or
- $\sigma = \sqrt{8^2 + 3^2 + 17^2} = \sqrt{362} \simeq 19$, if the Euclidean norm is used.

7.3 Effective Attack Surface

In general, the attack surface can be considered as an attribute of a service S. As discussed by Liu [Liu2008], that attack surface can be calculated based on the software characteristics of the service S, such as software complexity, web service chain steps, etc. But, instead of considering the out of the box attack surface, I would like to introduce the notion of *effective attack surface*, which characterizes the service in an operation environment.

Definition 7.1. The *Effective Attack Surface* attribute of a service S, denoted by σ_E represents the actual value of the attack surface of the service S as seen by an attacker in operation.

If a service is deployed out of the box as it is, then its Effective Attack Surface σ_E will be the same as the original attack surface value. On the contrary, if the same service is configured to operate within the boundary of a set of firewalls, then its effective attack surface will be less than its original one. Using the same line of thought, if the same service is protected by SCIT, then its effective attack surface will be reduced. In fact, I will show mathematically that it can be reduced and controlled by SCIT exposure window.

Since a service in operation can be upgraded, patched, or re-configured to satisfy new functionalities or security requirements, its effective attack surface will take on different values over time. These values of the effective attack surface can form a sequence of

discrete values $(\sigma_E)_{i=1,\dots,n}$. If SCIT is employed, then for each value in the $(\sigma_E)_{i=1,\dots,n}$ sequence, there will be a corresponding value W_O for exposure window. The values of the latter parameter will also form a sequence $(W_O)_{i=1,\dots,n}$.

7.4 Atomic Service

7.4.1 Attack Probability and Surface

Starting from the analytical expressions obtained for the IT-QoS parameters such as MTTSF and SA, I have shown that the attack probability P_A affects the variations of MTTSF and SA (refer to equalities 4.3, 4.7). With SCIT being the basis for protecting a service, the decrease in P_A makes MTTSF and SA increase, and reducing the active window W_O can decrease P_A. In previous discussion, the attack probability has been modeled based on the arrival of attacks. Actually, for each attack, taking into account the attack surface leads to the introduction of the probability that an attack targets the entry and exit points residing on the surface of service S. With the assumption that the two events of attack arrival and attack surface targeting are independent of each other, the following theorem captures this integration of the attack arrival and the surface attack.

Theorem 7.1. Let P_R denote the attack arrival probability, and P_S the probability of attacking the surface of service S. Let X be the random variable for the attack arrival events, with cdf $F_X(t)$. Then, the attack probability P_A is given by:

$$P_A = P_R P_S \quad (7.1),$$

where

$$P_R = \frac{1}{W_O}\int_0^{W_O} F_X(t)dt.$$

Proof.

By taking into account the attack surface of service S and assuming that the two events of attack arrival and attack surface targeting are independent of each other, the service protected by SCIT goes from state G to state A with probability:

$$P_A = P(X < Y).P_S.$$

With P_R given in Theorem 4.1, we obtain P_A:

$$P_A = \frac{P_S}{W_O}\int_0^{W_O} F_X(t)dt.$$

QED.

The expression (7.1) has the following interesting implications:

a) There are now two ways to reduce the attack probability: one by limiting the attack arrivals to reach the service's surface, which can be achieved using simple SCIT mechanism; and the other by reducing the attack surface, either via improving software quality attributes or hardened configuration. Combined together, these two ways to minimize the attack probability form what I termed as intrusion tolerance strategy along space-time dimensions.

b) According to Theorem 4.1, contracting P_R can be achieved by reducing the active window W_O. Therefore, with SCIT as a foundational mechanism, W_O can be used to compensate the expansion of P_S. This will constitute a practical way to overcome the drawback of widening the attack surface of service S, when the latter is enhanced over time with more functionalities in order to adapt to new environment and increasing users' demand.

Intuitively, the factor P_S in relation (7.1) would increase as the attack surface metric σ increases; thus, P_S can be modeled by a function G satisfying the following conditions:

$$\begin{cases} G(x) \text{ is defined for } x \in (0,\infty) \\ G(x) \text{is strictly monotonously increasing} \\ \lim\limits_{\sigma\to\infty} G(x) = 1 \\ \lim\limits_{\sigma\to 0} G(x) = 0 \end{cases} \quad (7.2).$$

Using the notion of effective attack surface σ_E, I can write:

$$G(\sigma_E) = P_R G(\sigma) \iff \sigma_E = G^{-1}\big(P_R G(\sigma)\big) \quad (7.3).$$

Since G is monotonously increasing, the inverse G^{-1} in (7.3) must exist and is also monotonously increasing. If SCIT is utilized, then we know that P_R can be expressed as a function F(W) of the SCIT window W such that:

$$\begin{cases} F(x) \text{ is defined for } x \in (0,\infty) \\ F(x) \text{is strictly monotonously increasing} \\ \lim\limits_{\sigma\to\infty} F(x) = 1 \\ \lim\limits_{\sigma\to 0} F(x) = 0 \end{cases} \quad (7.4).$$

Replacing P_R in (7.3) by the associated model function F characterized by (7.4) yields:

$$\sigma_E = G^{-1}\big(F(W).\,G(\sigma)\big) \quad (7.5).$$

From relation (7.5), we can state the following proposition:

Proposition 7.2. For a given value of the attack surface metric σ of a service, the effective attack surface σ_E will decrease along with W if SCIT is employed.

For a service without SCIT, it is clear that the effective attack surface $\sigma_E = \sigma$, by just letting $W_O = \infty$ in (7.5). For a fixed σ value of attack surface, the function $\sigma_E = G^{-1}(W_O)$ has the following characteristics:

$$\begin{cases} G^{-1}(x) \text{ is defined for } x \in (0,\infty) \\ G^{-1}(x) \text{is strictly monotonously increasing} \\ \lim\limits_{\sigma\to\infty} G^{-1}(x) = \sigma \\ \lim\limits_{\sigma\to 0} G^{-1}(x) = 0 \end{cases}$$

7.4.2 Exponential Distribution

If G(x) is modeled according to the exponential distributions of parameter λ_2, G(x) is given by:

$$G(x) = 1 - e^{-\lambda_2 x}.$$

With the Poisson attack arrival, Theorem 4.3 from chapter 4 gives F(x) as:

$$F(x) = 1 - \frac{1}{\lambda_A x}\left(1 - e^{-\lambda_A x}\right).$$

Then, the inverse function G^{-1} is:

$$G^{-1}(x) = \frac{1}{\lambda_2}\ln\left(\frac{1}{1-x}\right) \text{ for } x \in (0,1) \qquad (7.6),$$

and by applying (7.5), I obtain the effective attack surface:

$$\sigma_E = \frac{1}{\lambda_2} \ln\left(\frac{1}{1-\left(1-\frac{1}{\lambda_A W_O}\left(1-e^{-\lambda_A W_O}\right)\right)\left(1-e^{-\lambda_2 \sigma}\right)}\right) \qquad (7.7).$$

Note that modeling G by an exponential function is plausible with the assumption that the attackability of any surface unit is the same (i.e. the expected rate of attack occurrences per surface unit of measure is fixed to λ_2); this can be conditioned by the computation of the attack surface metric σ with appropriate weighting assignment to various resources exposed by the service in question.

As discussed in Chapter 4, P_R still decreases with W_O, hence I can arrive at the same conclusion as for the exponential distribution, i.e. σ_E decreases as W_O does.

How can I apply these results to QFITS? Let's say I want to design a service S with a desired value m for MTTSF. If I put the service S within a SCIT environment, then from the Semi-Markov process analysis, I know that MTTSF depends on the attack probability P_A. The latter value can be controlled by the attack surface metric σ and the exposure window W. If σ is given, then I can tune W to compensate σ and reduce P_A to a level that yield the desired MTTSF.

7.5 Impact on IT-QoS Expressions

With the introduction of the attack surface into the attack probability P_A, there is a need to revise the analytical expressions obtained in a previous chapter for the IT-QoS.

For generality, I go directly to the model of Weibull distribution for the attack arrival, since the exponential distribution model is one of its special cases, where the arrival rate is constant. In this case, the expressions for MTTSF, *SA*, and *SR* obtained in chapter 4 will become functions of the SCIT window W_O and the attack surface metric σ. For instance, the expression for MTTSF in chapter 4 will become:

$$\text{MTTSF} = \frac{H_G + P_R P_S H_A}{P_R P_S \bar{P_C}} \qquad (7.8).$$

If the exponential distributions are used for the model, then, according to Theorems 4.3 and 4.4 and Proposition 4.5, the elements of the two expressions above are:

$$\begin{cases} H_G = \dfrac{1}{\lambda_A} - \dfrac{1}{{\lambda_A}^2 W_O}(1 - e^{-\lambda_A W_O}) \\ H_A = \dfrac{1}{\lambda_F} - \dfrac{1}{{\lambda_F}^2 W}(1 - e^{-\lambda_F W}) \\ \mathrm{P_R} = 1 - \dfrac{1}{\lambda_A W_O}\left(1 - e^{-\lambda_A W_O}\right) \\ \overline{\mathrm{P_C}} = 1 - \dfrac{1}{\lambda_F W}\left(1 - e^{-\lambda_F W}\right) \\ \mathrm{P_S} = \mathrm{G}(\sigma) \end{cases} \qquad (7.9).$$

$G(\sigma)$ is added as a function of the attack surface metric σ; $G(\sigma)$ can follow some distribution satisfying the conditions (7.2), or an exponential distribution with parameter λ_2. In this case, it is clear that MTTSF and SA are multivariate functions f_1 and f_2 as denoted by:

$$\mathrm{MTTSF} = \mathrm{f_1}(\mathrm{W}, \lambda_\mathrm{A}, \lambda_\mathrm{F}, \sigma, \lambda_2) \quad , \text{and}$$

$$\mathrm{SA} = \mathrm{f_2}(\mathrm{W}, \lambda_\mathrm{A}, \lambda_\mathrm{F}, \sigma, \lambda_2).$$

7.6 Enabler Service

In Chapter 5, I have established that we can control the attack probability P_A to service S via the cleansing probability P_{C1} of the enabler service S_1 according to the relation (5.1) listed below:

$$\mathrm{P_A} = \overline{\mathrm{P}}_{\mathrm{C1B}}.$$

On the other hand, with W_1 being the SCIT window of service S_1 and λ_F the parameter of the exponential function for the attack compromise, the value of $\overline{\mathrm{P}}_{\mathrm{C1}}$ is given in the proof of Theorem 4.2 in Chapter 4 as a function of W_1:

$$\overline{\mathrm{P}}_{\mathrm{C1B}} = \frac{1}{\mathrm{W_1}} \int_0^{\mathrm{W_1}} \mathrm{F_X}(\mathrm{t})\mathrm{dt} = \mathrm{F_{S1}}(\mathrm{W_1}).$$

In the case of Poisson model, the function $F_{S1}(W_1)$ becomes:

$$F_{S1}(W_1) = 1 - \frac{1}{\lambda_F W_1} e^{-\lambda_F W_1}.$$

Combining these two relations just above, I can deduce that:

By introducing the surface attack into P_A in the above equality, I obtain:

$$P_A = \overline{P}_{C1B} \Leftrightarrow F(W).G(\sigma_E) = F_{S1}(W_1)$$

$$\Leftrightarrow \sigma_E = G^{-1}\left(\frac{F_{S1}(W_1)}{F(W)}\right)$$

Now, let's determine the variation of the effective attack surface with respect to W_1 by calculating the partial derivative of the function in (7.9).

$$\frac{\partial \sigma_E}{\partial W_1} = \frac{\frac{\partial F_{S1}}{\partial W_1}}{F(W).\frac{\partial G}{\partial W_1}(G^{-1}\left(\frac{F_{S1}(W_1)}{F(W)}\right))}.$$

Thanks to the characteristics of F, F_1, and G as specified by (7.2) and (7.4), we have:

$$\forall x \in (0,\infty), \quad \frac{\partial F_{S1}}{\partial W_1} > 0, \text{and } \frac{\partial G}{\partial W_1} > 0; \text{hence: } \frac{\partial \sigma_E}{\partial W_1} > 0.$$

From the signs of the derivatives above, I can state the following proposition.

Proposition 7.3. The effective attack surface metric will decrease if the SCIT window W_1 of the enabler service S_1 decreases.

In the particular case where F, F_{S1}, and G are modeled by exponential distributions, I can express the effective attack surface σ_E of service S in a composite with enabler service S_1 as:

$$\sigma_E = \frac{1}{\lambda_2} \ln\left(\frac{1}{1 - \frac{1 - \frac{1}{\lambda_F W_1} e^{-\lambda_F W_1}}{1 - \frac{1}{\lambda_A W}(1 - e^{-\lambda_A W})}}\right)$$

$$\Leftrightarrow \quad \sigma_E = \frac{1}{\lambda_2} \ln\left(\frac{1 - \frac{1}{\lambda_A W}\left(1 - e^{-\lambda_A W}\right)}{\frac{1}{\lambda_F W_1} e^{-\lambda_F W_1} - \frac{1}{\lambda_A W}(1 - e^{-\lambda_A W})}\right) \qquad (7.10).$$

7.7 Diversity

Diversity has been one of the mechanisms to provide intrusion tolerance. For instance, in the SITAR architecture, the voting algorithm partly relies on the web servers that provide the same functionalities implemented by different products such as Apache Web Server or Microsoft Internet Information Services (IIS) [Wan2003a]. According to [Hua2011], service resilience can be achieved by means of diversity and service pro-active recovery; this diversity can be realized by considering the multi-tier architecture, where various versions and configurations are used to implement each layer, from hardware, operating systems, middleware, web servers, and software applications. Thus, multiple configurations can be devised, possibly leading to a combinatorial number of different versions of the same service's functionalities. For instance, with four flavors of Linux (Red Hat, Ubuntu, Debian, and Fedora), two versions of Windows OS (Windows 2003 and Windows 2008), and two web servers products (Apache and Microsofft IIS), one can easily build different configurations to host a web site. However, it has been pointed out in [Hua2011] that there is some cost associated with managing the different

configurations of a service. Since the proposed scheme affects all layers of the technology stack used to realize the service, this type of management has to handle the patching, upgrading, compatibility matrix, and inter-dependency between different versions of the components comprising the service. W$W_O$ith the assumptions that Cloud service providers have a large and diverse array of computing resources, diversity can be realized in a transparent manner to the Cloud users in the case of C-SCIT, a cloud-based version of SCIT [Ng2011].

Let's go back to the case of an atomic service protected by SCIT combined with diversity. The reasoning in this section can be applied to any pro-active and deterministic recovery intrusion tolerance scheme with an algorithm that keeps rotating the versions of the service. I will describe how to perform the quantitative evaluation of such a system, with the goal of ensuring certain level of IT-QoS.

Let:

- $\{s_i \mid i=1,..,V\}$ be the set of V different configurations of the same service S.
- (σ) be the random variable whose state space is the set $\{\sigma_i \mid i=1,..,V\}$ where σ_i is the attack surface metric associated with the configuration s_i of service S.
- P_i be the probability that the configuration s_i is chosen by the rotation algorithm.

Then, the expected value of the attack surface metric σ is:

$$\overline{\sigma} = E[\sigma] = \sum_{i=1,..,V} P_i \, \sigma_i \qquad (7.11).$$

In the special case where the employed rotation algorithm is not biased towards any specific configuration, each configuration of the service S can be chosen with equal probability 1/V. Thus, the expected value of σ becomes:

$$\overline{\sigma} = \sum_{i=1,..,V} \frac{1}{V}\,\sigma_i \qquad (7.12).$$

The diversity configured into the operation of a service creates a random event to each attack arrival. Usually, an attack tries to exploit specific entry or exit points of a surface, or a vulnerability that exists in a specific implementation. Due to this reason, the chance that an attack matches a service configuration is only 1/V, V being the number of configurations being used by the rotation algorithm. Therefore, I can write:

$$\frac{1}{V}G(\sigma_{min}) \le P_S \le \frac{1}{V}G(\sigma_{max}) \qquad (7.13),$$

where

$$\sigma_{min} = \min_{i=1,..V}\{\sigma_i\} \text{ and } \sigma_{max} = \max_{i=1,..V}\{\sigma_i\}.$$

The inequalities in (7.13) clearly show that the surface attack probability becomes smaller and smaller as V increases, and:

$$\lim_{V\to\infty} P_S = 0.$$

If the expected value of σ in (7.12) is used, then P_S can be written as:

$$P_S = \frac{1}{V}\,G(\overline{\sigma}) \qquad (7.14).$$

From (7.14), I can derive the effective attack surface when diversity with V configurations is employed to protect service S:

$$P_S = G(\sigma_E) = \frac{1}{V}\ G(\bar{\sigma}) \Leftrightarrow \sigma_E = G^{-1}[\frac{1}{V}G(\bar{\sigma})] \quad (7.15).$$

According to (7.15), the following proposition can be derived:

Proposition 7.4. Let S be a service protected by SCIT. Given the attack surface of a service S, the attack probability P_S and the effective attack surface σ_E of that surface decreases as the number of variants V increases.

In particular, if G represents the exponential distribution with parameter λ_2, then P_S and σ_E will take the following expressions:

$$P_S = \frac{1}{V}\ \left(1 - e^{-\lambda_2 \bar{\sigma}}\right) \qquad (7.16)$$

$$\sigma_E = \frac{1}{\lambda_2}\ \ln\left(\frac{1}{1 - \frac{1}{V}(1 - e^{-\lambda_2 \bar{\sigma}})}\right)$$

$$\sigma_E = \frac{1}{\lambda_2}\ \ln\left(\frac{V}{V + e^{-\lambda_2 \bar{\sigma}} - 1}\right) \quad (7.17).$$

The general relation (7.15) and the special case one in (7.17) provide analytical expressions showing that the effective attack surface can be diminished by rotationally exposing different configurations of the service. From the system design perspectives, the rotation algorithm of various service configurations can be viewed as an intrusion tolerance strategy integrating space and time dimensions.

7.8 Summary

In this chapter, I have proposed to introduce the dynamicity of the attack surface of a service, whose implementation will most likely evolve over time, due to feature enhancements, software patches and adapted configurations. This dynamicity is taken further when the attack surface concept is integrated with the continuous rotational cleansing and replacement of instances of the same service throughout the operation of that service. Such a combined strategy opens new possibilities to system designer to:

- Compensate a given attack surface by smaller exposure window;
- Reduce the attack surface of a target service by reducing the exposure window of its enabler service in a service orchestration;
- Complement the cleansing rotation by having diverse configurations of the service.

These implications have been supported by mathematical reasoning keyed on the attack probability identified and analyzed in Semi-Markov Process in previous chapters. Moreover, the two major IT-QoS parameters, namely MTTSF and SA have been expressed in terms of the probabilities P_C, P_R, and P_S for cleansing, attack arrival, and successful attack to a surface respectively. From my analysis, I can infer that MTTSF and SA depend on the attack surface metric σ and the exposure window W. Tuning these two control parameters will help achieving the desired values for MTTSF and SA, using the analytical expressions obtained in the chapters from 4 through 6.

CHAPTER 8 - DESIGNING SERVICES WITH MULTI-LEVEL IT-QOS

8.1 Introduction

Using Semi-Markov Process to model the services protected by the recovery-based intrusion tolerance, I have obtained analytical expressions of the IT-QoS of interest in terms of the control parameters in Chapters 4-6. In addition, the previous chapter introduces the attack surface to the analysis model as a factor influencing the realization of the IT-QoS parameters of a service. The goal of this Chapter is to apply those results in designing services with desired levels of IT-QoS within the framework of QFITS. For the IT-QoS, I will focus on the attributes of availability and reliability. The design will leverage the extension of the UML Profile described in chapter 3. For a service protected by SCIT or a variation of SCIT, the problem is how to design a service that can guarantee probabilistically a set of IT-QoS values, given the environment parameters. These consist of:

- Service cleansing time, denoted by T_C, which will depend on the memory footprint of the service. If virtualization is used, then T_C will be influenced by the time to load the service image on the hypervisor.
- Service attack surface metric σ, which can be evaluated based on one of the schemes described in [How2001, Liu2008, Heu2010].

- Attack arrival rate λ_A, which can be estimated based on historical data, if Poisson model is used. The usage of this attack arrival rate is based on the assumption that the arrival distribution is exponential. If the enterprise does not have historical data for intrusion rates, then public reports such as Verizon DBIR [Ver2013] could be used as the starting point for this estimation.
- Attack residency rate, or compromise rate λ_F, which is constrained by the attack arrival rate above and the exposure window. For simplicity, λ_F can be estimated by λ_A for the reason that during the time lapse of an exposure window, the residency rate in one service instance cannot exceed the arrival rate.
- Mean sojourn times at the states of the Semi-Markov Process, namely the value for the vector H. These times are usually estimated based on historical data. In Chapter 4, I have shown that closed forms can be obtained for the mean sojourn times in the case of Poisson and Weibull models.

8.2 IT-QoS Design for an Atomic Service

For the design of an atomic service with multi-level of IT-QoS, two cases are considered: a) The first one is called static configuration which is usually useful as an initial design and deployment of a service; b) the second one handles dynamic configuration to allow the service to adapt to changing operating conditions in order to maintain the intrusion tolerance guarantees. While we assume that computing resources can be allocated accordingly for the initial deployment of a service, the dynamic configuration has to be considered in two scenarios: one where computing resources can be expanded to satisfy the demand of the reconfiguration, the other where the resources are given and fixed.

In particular, given a set of levels for MTTSF and SA, the question is how to design a service guaranteeing those levels. Note that the computation of the S-Availability and S-Reliability can be derived from the values of MTTSF and SA.

8.2.1 Static Configuration

As shown in chapter 7, I have obtained a relation between the exposure window and the two IT-QoS MTTSF and SA can be expressed as multivariate functions with variables the exposure window W, the attack surface metric σ, the attack arrival parameters . The exact expressions of f_1 and f_2 depend on which distribution function is used to model the attack arrival and behavior.

Let U = (U_1, ..., U_k) and R = (R_1, ..., R_k) be *k* desired values for MTTSF and SA respectively. Let's denote by N = (N_1, N_2, ..., N_k) the array of number of nodes needed to achieve k levels of MTTSF and SA. Assume that the attack arrival is Poisson, and exponential distributions are modeled for the sojourn times, as well as the surface attack. The algorithm to configure a service S with MTTSF levels given by U is summarized in the following table. The idea is to solve the value for the exposure window W that is required to satisfy the desired level of MTTSF. Given that the function F_1 is monotonously decreasing with respect to W, the equation in step 5 below should yield a solution for W. Recall from Chapter 6, that the expressions for MTTSF and SA in the case of a simple service protected by SCIT are given by:

$$\text{MTTSF} = \frac{H_G + P_R P_S H_A}{P_R P_S \bar{P_C}}, \text{and}$$

$$\text{SA} = \frac{H_G + P_R P_S H_A}{H_G + P_R P_S H_A + P_R P_S \bar{P_C} H_F}.$$

If Poisson model is used for attack arrival and exponential distributions for sojourn times, then the elements in the above formulas can be expressed in terms of W and the parameters of the distributions, as in (4.10), (4.11), and (4.13).

Table 8.1. Static Configuration for MTTSF

configureMTTSF(U, σ, T_C, λ_A, λ_F, λ_2)
1. **for** each i from 1 to k
2. (W_i, N_i) = **computeMTTSF(U_i, σ, λ_A, λ_F, λ_2)**
3. **end for**
4. Return ((W_1,N_1), ... (W_k, N_k)).

computeMTTSF(u, σ, λ_A, λ_F, λ_2)
5. Solve W in the equation:
u = $F_1(W, \lambda_A, \lambda_F, \sigma, \lambda_2)$.
6. $W = \lfloor W \rfloor$
7. Using (3.7), compute the number of nodes N based on the solution W obtained in step 5:
$$N = \left\lceil \frac{2T_C}{W} \right\rceil + 2$$
8. Return (W, N).

The steps to configure a service S with SA levels given by R are very similar to the ones used to configure MTTSF levels, except the function used in step 5.

Table 8.2. Static Configuration for SA

configureSA(R, σ, T_C, λ_A, λ_F, λ_2)
1. **for** each i from 1 to k
2. (W_i, N_i) = **computeSA(R_i)**
3. **end for**
4. Return ((W_1,N_1), ... (W_k, N_k)).

computeSA(r)
5. Solve W in the equation:

$$r = f_2(W, \lambda_A, \lambda_F, \sigma, \lambda_2).$$

6. $W = \lfloor W \rfloor$
7. Using (3.7), compute the number of nodes N based on the solution W obtained in step 5:

$$N = \left\lceil \frac{2T_C}{W} \right\rceil + 2$$

8. Return (W, N).

The next question is how to reconcile if Algorithms 8.1 and 8.2 output two different sets of values for the exposure window and number of nodes. Recall that the effect of the exposure window is not contradictory for MTTSF and SA, i.e. decreasing or increasing the exposure window will produce the same desirable effect on both MTTSF and SA. Therefore, the reconciliation simply consists of choosing the lowest value of the exposure window with its associated number of nodes for each level of IT-QoS. Algorithm 8.3 below shows how to configure an atomic service with desired levels of MTTSF and SA.

Table 8.3. Static Configuration for (MTTSF, SA).

configure(U,R, σ, T_C, λ_A, λ_F, λ_2)

1. (W_U,N_U) = **configureMTTSF(U, σ, T_C, λ_A, λ_F, λ_2)**
2. (W_R,N_R) = **configureSA(R, σ, T_C, λ_A, λ_F, λ_2)**
3. **for** each i from 1 to k
4. W_i = min($W_{U,I}$, $W_{R,I}$)
5. $N_i = \left\lceil \frac{2T_C}{W_i} \right\rceil + 2$
6. **end for**

7. Return ((W_1,N_1), ... (W_k, N_k)).

8.2.2 Numerical Examples

In this example, let the values of the parameters required in the algorithms be as followed:

$\lambda_A = \lambda_F = 10^{-3}$, $\lambda_2 = 1$, $T_C = 5$, and $\sigma = 10000$. I also assume that the probability on the attack surface can be expressed as the exponential distribution with parameter $\lambda_2 = 1$:

$$P_S = 1 - e^{-\lambda_2 \sigma}$$

Table 8.4. Example - Static Configuration for (MTTSF, SA).

Level	MTTSF	W_U	SA	W_R	W	N
1	200000	10	0.5	4555	10	3
2	250000	8	0.6	3230	8	4
3	500000	4	0.7	2273	4	5
4	1000000	2	0.8	1526	2	7
5	2000000	1	0.9	883	1	12

8.3 Dynamic Configuration

8.3.1 Algorithm

It has been pointed out in [Ng2010] that the during the initial design of a service, the system architect has the leeway to perform system sizing such that computing resources, including hardware nodes, memory and cache configuration, number of virtual machines, are able to different levels of IT-QoS that the service provider wants to furnish. But, once the system is in operation, there may be constraints due to the finite amount of resources that determine the adaptability of the service in the eventuality of changing conditions, especially in the scenario that the monitoring system detects an increase in the attack rate. However, if a service is deployed in a cloud environment, and if the available of resources in the cloud is assumed to be much larger than the initial demand of the service to the point that they appear infinite, then the readjustment of the service does not present a big challenge. In this case, the dynamic configuration is tantamount to the recalculation

of the algorithms listed in Tables 8.1 and 8.2 with new values for the attack arrival and compromise parameters λ_A and λ_F respectively. For this reason, I will focus on the case when the computing resources are fixed after the deployment of the computing resources services. According to [Ng2010], the constraint is the maximum number of nodes N_{max} that have been provisioned for the SCIT system.

First, the architecture has to be augmented to have the new module called Dynamic Reconfiguration to be invoked by the Central Controller. The UML collaboration diagram in Figure 8.1 from [Ng2010a] depicts the interaction of the components within the architecture to support this reconfiguration to maintain the same level of S-Reliability. The alert sent by the Monitor service to the Central Controller that the attack arrival rate has changed will trigger the latter to compute new values for the exposure windows and SCIT nodes. The Service Registry shown in the diagram is used by the Central Controller to keep track of the parameters of the services to be protected, including current IT-QoS values, exposure window, and number of nodes in the cluster. After the new values are computed and used to update the cleansing algorithm executed by the Central Controller, they will be persisted in the Service Registry.

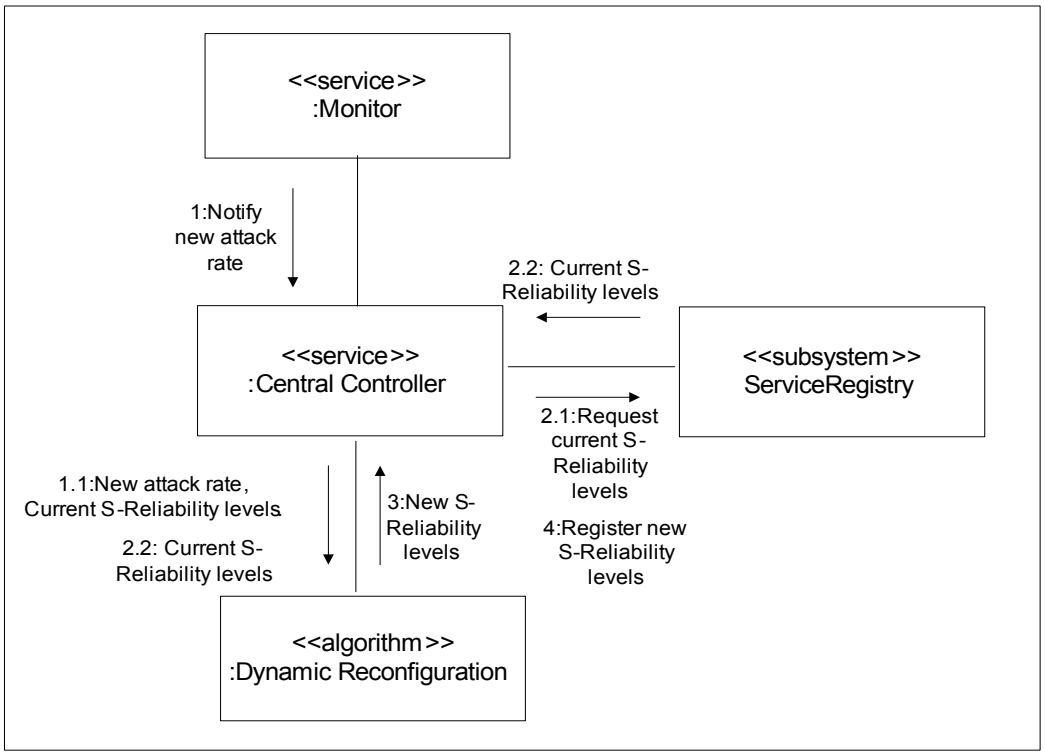

Figure 8.1. Reconfiguration of S-Reliability [Ng2010a].

The algorithms for reconfiguring the services to maintain the same levels of MTTSF and SA make use of the same functions computeMTTSF() and computeSA(). In the following listing, let's denote by:

- **Wc** = (Wc_1, W_{C2}, …, W_{Ck}): current exposure window values
- **Nc** = (N_{C1}, N_{C2}, …, N_{Ck}): current numbers of nodes
- $\mathbf{U_C}$ = (R_{C1}, R_{C2}, …, R_{Ck}): current levels of MTTSF
- $\mathbf{R_C}$ = (R_{C1}, R_{C2}, …, R_{Ck}): current levels of SA
- N_{max}: total number of nodes available for SCIT
- N_A: remaining available nodes.

Since the total number of nodes is capped at N_{max}, the algorithms have to check whether the new values for the number of nodes exceed the capacity, as exhibited in line 5. If the new value of number of nodes does not exceed the capacity, then the new values for exposure windows and SCIT nodes will be accepted. Otherwise, the algorithms just keeps the current values for the exposure window and number of nodes, as shown in line 8; in this case, the adaptation cannot be done.

Table 8.5. Dynamic reconfiguration Algorithm for MTTSF.

reconfigureMTTSF(U, σ, T_C, λ_A, λ_F, λ_2, N_{max}, Wc, Nc, U_C)

1. $N = \sum_{i=1}^{K} N_{Ci}$. // total number of nodes used so far
2. $N_A = N_{max} - N$
3. **for** each i from 1 to K
4. (W_i, N_i) = **computeMTTSF(U_i, σ, λ_A, λ_F, λ_2)**
5. **if** $(N_A + N_{Ci} - N_i > 0)$ **then**
6. $N_A = N_A + N_{Ci} - N_i$
7. **else**
8. $W_i = W_{Ci}$; $N_i = N_{Ci}$; $U_i = U_{Ci}$
9. **endif**
10. **end for**
11. **return** (U_1,…, U_k) and ((W_1, N_1), … , (W_k,N_k)).

The reconfiguration algorithm for SA follows the similar steps as shown below.

Table 8.6. Dynamic reconfiguration Algorithm for SA.

reconfigureSA(R, σ, T_C, λ_A, λ_F, λ_2, N_{max}, Wc, Nc, R_C)

1. $N = \sum_{i=1}^{K} N_{Ci}$. // total number of nodes used so far
2. $N_A = N_{max} - N$
3. **for** each i from 1 to K
4. (W_i, N_i) = **computeSA(R_i, σ, λ_A, λ_F, λ_2)**
5. **if** $(N_A + N_{Ci} - N_i > 0)$ **then**
6. $N_A = N_A + N_{Ci} - N_i$
7. **else**
8. $W_i = W_{Ci}$; $N_i = N_{Ci}$; $U_i = U_{Ci}$
9. **endif**
10. **end for**
11. **return** ($R_1, \ldots, R_k$) and ((W_1, N_1), … , (W_k,N_k)).

The algorithms 8.3 and 8.4 are rudimentary. Some more sophisticated schemes can be designed to either give priority for allocation to highest QoS levels, or readjust evenly the resources among all levels.

8.4 Designing a Service with (SCIT+IDS)

8.4.1 Algorithm

When a service is secured by the combination (SCIT+IDS), there are three control parameters in play: detection probability, false positive probability, and exposure window. Selecting the values for the detection and false positive probabilities can be performed using the Receiver Operating Characteristic (ROC) curve or some cost optimization [Gaf2001, Nag2011]. Once these values are selected, they will be fed into the configuration algorithm. The core function of the algorithm listed below depends on the analytical expressions (6.14) and (6.21) derived for MTTSF and SA, in the case

where the service is protected by the combination of SCIT and IDS. In these two expressions, I replace P_A by the product ($P_R P_S$) to account for the attack surface metric:

$$\text{MTTSF} = \frac{H_G + P_R P_S H_A + P_{FP} H_{FP} + P_R P_S \bar{P_C} P_D H_D + P_R P_S \bar{P_C} P_U H_U}{P_R P_S \bar{P_C} P_U \overline{P_{UC}}} \quad (8.1),$$

$$\text{SA} = \frac{H_G + P_R P_S H_A + P_{FP} H_{FP} + P_R P_S \bar{P_C} P_D H_D + P_R P_S \bar{P_C} P_U H_U}{H_G + P_R P_S H_A + P_{FP} H_{FP} + P_R P_S \bar{P_C} P_D H_D + P_R P_S \bar{P_C} P_U H_U + P_R P_S \bar{P_C} P_U \overline{P_{UC}} H_F} \quad (8.2).$$

According to the discussion in Chapters 4 and 6, MTTSF and SA can be expressed as functions f_3 and f_4 respectively, with appropriate assumptions about the Poisson model and exponential distribution; f_3 and f_4 are just formulations derived from (8.1) and (8.2) above:

$$\text{MTTSF} = f_3 (W, \lambda_A, \lambda_F, \sigma, \lambda_2, P_D, P_{FP}), \quad \text{and}$$

$$\text{SA} = f_4 (W, \lambda_A, \lambda_F, \sigma, \lambda_2, P_D, P_{FP}).$$

Thus, the algorithms for configuring the service with desired levels for MTTSF and SA are listed below.

Table 8.7. MTTSF Configuration with (SCIT+IDS).

configureMTTSF2(U, σ, T_C, λ_A, λ_F, λ_2, P_D, P_{PF})

for each i from 1 to k
2. (W_i, N_i) = **computeMTTSF2(U_i)**
3. **end for**
4. Return ((W_1,N_1), ... (W_k, N_k)).

computeMTTSF2(u)
5. Solve W in the equation:

$$u = \lfloor f_3(W, \lambda_A, \lambda_F, \sigma, \lambda_2, P_D, P_{FP}) \rfloor$$

6. Using (3.7), compute the number of nodes N based on the solution W obtained in step 5:

$$N = \left\lceil \frac{2T_C}{W} \right\rceil + 2$$

7. Return (W, N).

Table 8.8. SA Configuration with (SCIT+IDS).

configureSA2(R, σ, T_C, λ_A, λ_F, λ_2, P_D, P_{PF})

for each i from 1 to k
2. (W_i, N_i) = **computeSA2(R_i)**
3. **end for**
4. Return ((W_1,N_1), ... (W_k, N_k)).

computeSA2(r)
5. Solve W in the equation:

$$r = \lfloor f_4(W, \lambda_A, \lambda_F, \sigma, \lambda_2, P_D, P_{FP}) \rfloor$$

6. Using (3.7), compute the number of nodes N based on the solution W obtained in step 5:

$$N = \left\lceil \frac{2T_C}{W} \right\rceil + 2$$

7. Return (W, N).

8.4.2 Numerical Examples

In addition to the same parameter values used in 8.2.2, namely

$\lambda_A = \lambda_F = 10^{-3}$, $\lambda_2 = 1$, $T_C = 5$, $\sigma = 10000$, and $\lambda_2 = 1$, I need to assign values for the detection and false positive probabilities: $P_D = 0.9$, and $P_{FP} = 0.05$.

The values in the table below reveal that (SCIT+IDS) can dramatically increase the resilience of the service, although the attack arrival has been increased by an order of magnitude. As a result of the effectiveness of the protection provided by (SCIT+IDS), it requires only 3 nodes to provide different levels of MTTSF, in this particular scenario with the given parameters. Note that 3 is the bare minimum number of nodes in a SCIT architecture.

Table 8.9. Example - Static Configuration for MTTSF under (SCIT+IDS).

Level	MTTSF	W	N
1	200000	1004	3
2	250000	836	3
3	500000	511	3
4	1000000	332	3
5	2000000	223	3

8.5 Designing a Service with (SCIT+Diversity)

8.5.1 Algorithm

If SCIT is combined with diversity as in the moving defense target scheme proposed in [Hua2011], then the control parameters are the exposure window, and the number of variants N. There are three scenarios, all of which require the service to satisfy given levels of MTTSF.

- Scenario 1, where the exposure window W is fixed, and the number of variants or versions V need to be calculated.
- Scenario 2, where V is fixed, and W has to be determined.
- Scenario 3, which asks for finding an optimal combination of W and V. This would need a cost structure in order to formulate a cost optimization problem.

In this section, I will consider Scenarios 1 and 2 only. Let denote by σ_0 the original attack surface metric, and $V = (V_1, V_2, ..., V_k)$ be the array containing k values for the needed number of variants.

Table 8.10. MTTSF Configuration with (SCIT+Diversity) and fixed W.

configureMTTSF3(U, W, σ_0, T_C, λ_A, λ_F, λ_2)
1. **for** each i from 1 to k
2. V_i = **computeMTTSF3(U_i, W, σ_0, T_C, λ_A, λ_F, λ_2)**
3. **end for**
4. Using (3.7), compute the number of nodes N based on the solution W obtained in step 5:

$$N = \left\lceil \frac{2T_C}{W} \right\rceil + 2$$

4. Return ((W_1,N), ... (W_k, N)).

computeMTTSF(u, W, σ_0, T_C, λ_A, λ_F, λ_2)
5. Solve **σ** in the equation:

$$u = f_1(W, \lambda_A, \lambda_F, \sigma, \lambda_2)$$

6. $V = \left\lceil \frac{\sigma_0}{\sigma} \right\rceil$
7. Return V.

For Scenario 2 where V is fixed, the algorithm is sketched below.

Table 8.11. MTTSF Configuration with (SCIT+Diversity) and fixed V.

configureMTTSF4(U, V, σ_0, T_C, λ_A, λ_F, λ_2)
1. **for** each i from 1 to k
2. (W_i, N_i) = **computeMTTSF4(U_i, σ_0/V, T_C, λ_A, λ_F, λ_2)**
3. **end for**
4. Using (3.7), compute the number of nodes N based on the solution W obtained in step 5:

$$N = \left\lceil \frac{2T_C}{W} \right\rceil + 2$$

4. Return ((W_1,N_1), ... (W_k, N_k)).

computeMTTSF4(u, σ_0/V, T_C, λ_A, λ_F, λ_2)
5. Solve W in the equation:

$$u = f_1\left(W, \lambda_A, \lambda_F, \frac{\sigma_0}{V}, \lambda_2\right).$$

6. Using (3.7), compute the number of nodes N based on the solution W obtained in step 5:

$$N = \left\lceil \frac{2T_C}{W} \right\rceil + 2$$

7. Return (W,N).

8.5.2 Numerical Examples

In this example, I will reuse some of the parameter values in 8.2.2, namely: $\lambda_A = \lambda_F = 10^{-3}$ and $T_C = 5$. The original attack surface is $\sigma = 10000$, and the parameter for the attack on the surface is changed to: $\lambda_2 = 0.01$.

1) As an illustration of Scenario 1, let W = 10 time units. Hence, the number of needed nodes is just 3.

Table 8.12. Example - Configuration for MTTSF with (SCIT+Diversity) and W=10.

Level	MTTSF	W	V	N
1	202000	10	16	3
2	250000	10	62	3
3	500000	10	195	3
4	1000000	10	446	3
5	2000000	10	946	3

2) For Scenario 2, let V = 50 variants.

Table 8.13. Example - Configuration for MTTSF with (SCIT+Diversity) and V=50.

Level	MTTSF	W	V	N
1	202000	12	50	3
2	250000	9	50	4
3	500000	4	50	5
4	1000000	2	50	7
5	2000000	1	50	12

8.6 Designing S-Reliable Composite Service

A composite service is constructed from other services, which can be atomic or even composite themselves. Technologies such as BPEL (Business Process Execution Language) exist today to facilitate service construction. They are essentially high-level

programming languages with logic constructs (if-then-else, for loop, parallel branching, error handling, etc.) operating at the service level, and orchestrating services playing the role of "module" to create a "program", that is a composite service. If a service S is considered as the top parent, then the services making S will form a tree, in which all direct children of a node must have the same mode of execution, either parallel or serial. Figure 8.2 illustrates a service S composed of seven services: the children S_1, S_2 and S_3 of S are all executed in parallel; the children S_4, S_5 and S_6 of the parent S_2 are serial; S_7 and S_8 of S_3 are parallel; finally, S_1 has no children.

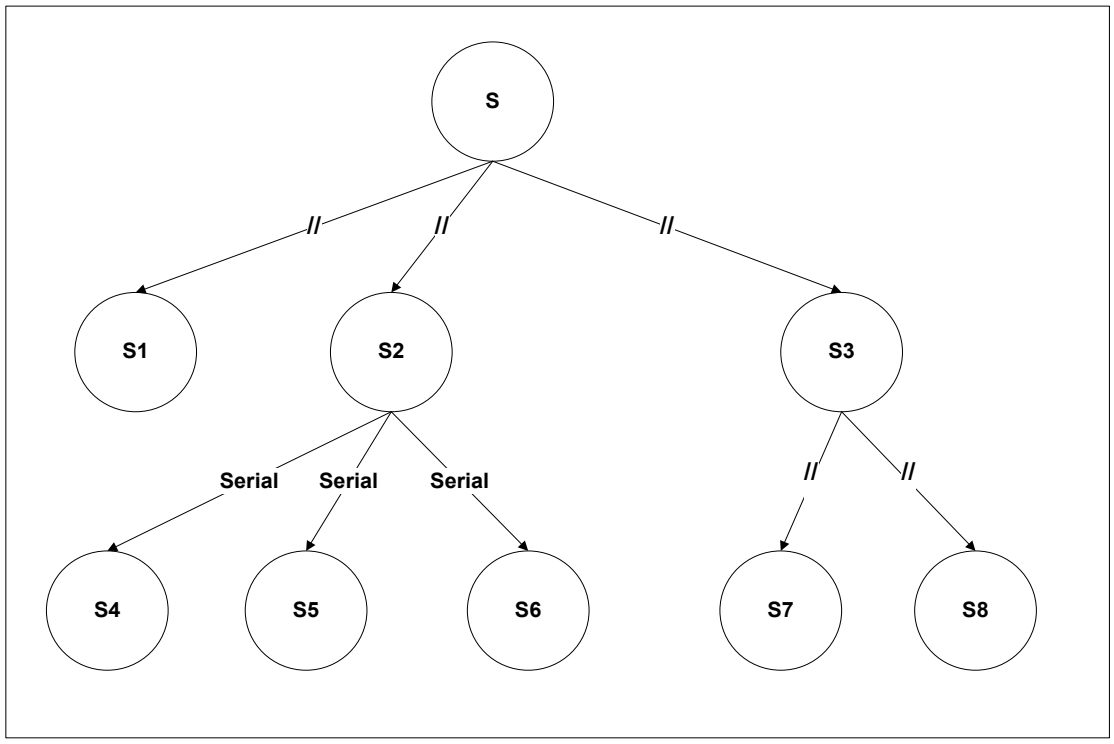

Figure 8.2. Service Composition Tree.

Formulas to compute the reliability of a system or service based on the reliabilities of the components are well-known in the theory of reliability. Given a service S having m direct service children S_i (i=1,...,m) with respective reliabilities R_i, the reliability R of S is given in [Ros2009, Men2002]:

$$SR = \begin{cases} \prod_{i=1}^{m} SR_i & if\ S_i\ are\ serial \\ 1 - \prod_{i=1}^{m} (1 - SR_i) & if\ S_i\ are\ parallel \end{cases} \quad (8.1)$$

Before applying these expressions to compute the S-Reliability of the same service S, a mapping between the logical constructs and the determination for applying the parallel versus serial formula.

Table 8.14. Mapping between Orchestration Logic and Reliability Formulas.

Logical Construct	BPEL	Formula	Notes
SERIAL	<sequence>	Serial	Sequential statements are invocations of services.
IF-THEN-ELSE	<switch>	$p_1SR_1+p_2SR_2$ + ..+p_kSR_k	For a given input, only one branch is executed.
LOOP ITERATION	<while>	Serial	Same services are executed over and over.
ASYNCHRONOUS BRANCHES	<flow>	Serial	Each branch consists of an asynchronous task. Although the branches are executed in

			parallel, the parent service requires that all of them succeed. Hence, the Serial Formula needs to be applied.
SELECT ONE OF THE BRANCHES	<pick>	Parallel	Only one branch is required.

Two problems are considered:

1) Problem SR-1: Given the services comprising the composite C, and their S-Reliabilities, how can we decide whether those services can be selected in order to satisfy S-Reliability of C? Thanks to the well-known formulas and the mapping table above, it is not difficult to calculate the resulting S-Reliability of C. A tree like the one in Figure 8.2 has to be built for service C. Then, the S-Reliability computation can be performed recursively by going down the nodes and branches of the tree and applying appropriate formulas given in Table 8.5. Starting from levels of S-Reliabilities of services S_i, selection can be done among those levels such that the computed S-Reliability satisfies the requirement of C. The following steps will be performed:

- Build the tree as the one shown in Figure 8.2 by applying the mapping table 8.14 to determine whether a path from a parent node to a child node is parallel or serial;
- Traverse the tree from the leaves to the root;

- At each level, compute the S-Reliability of the composite service represented by applying the appropriate formulas given in (8.1).

2) Problem SR-2: In this problem, services S_i are designed at the same time as the composite service C. Thus, how can the design proceed? Unlike the bottom-up approach as in the problem SR-1, a top-down approach starting from S and traversing down to the services in the tree can be done. One heuristic approach is to just give to each comprised service S_i the same S-Reliability such that the resulting S-Reliability is the one required by S. From those assigned values for the S-Reliabilities of S_i, the corresponding exposure windows and necessary SCIT nodes will be derived. A more optimal approach can be investigated in the future.

8.7 Summary

In this chapter, I have presented algorithms, both static and dynamic, that can be used to configure a service in order to satisfy and guarantee probabilistically the desired IT-QoS parameters. The core of these algorithms is to solve equations based on the analytical expressions obtained in previous chapters. What I have not explored is whether there is any computational issues in solving these equations numerically. However, given that the solutions are rounded either up or down to an integer, thus, decimal precision is not required, perturbation in the data will not likely impact the solutions.

CHAPTER 9 - CLOUD-BASED SCIT

9.1 Introduction

The characteristics of elasticity, scalability, rapid provisioning, and usage metering have made Cloud Computing a technology with increasing momentum in the Information Technology world. Thanks to its multi-tenancy capability, shared computing resource pools including CPU time, memory, storage, and cost effectiveness, Cloud Computing has allowed organizations to migrate and deploy their applications to cloud environments, thus eliminating the needs of building and maintaining large data centers and system administration staff. Moreover, the adoption of Cloud Computing takes origin from virtualization technologies, increased available network bandwidth, standard network protocols that have been becoming ubiquitous in the Internet, and web services' APIs. But, with the advent of applications running in the Cloud, come security concerns, which include data confidentiality, integrity, and availability. It is known that cloud service providers always strived to protect their assets utilized by the cloud consumers from malicious intrusions. However, with the distributed nature of cloud computing to be provided over networks and open protocols, cloud services and applications still expose vulnerabilities, although intrusion prevention and detection were in place. Therefore, intrusion tolerance should be part of the solution in the defense-in-depth strategy for cloud applications and services.

It was pointed out in [Buy2010, Kea2009] that utilizing resources and services from more than one cloud provider will increase the resilience and QoS guarantee. This motivates the use of the InterCloud or Cloud of clouds, which has the capability to coordinate and make available the combined resources from multiple cloud environments. The Cloud Exchange proposed in [Buy2008] is essentially a market place with standardized interfaces to facilitate buying and selling services from participating cloud providers in a programmatic fashion. InterCloud can be used to ensure data integrity and confidentiality when the persistent store is implemented using by Cloud storage; theories and protocols have also been developed to support the fragmentation and assembly of data dispersed in multiple cloud persistent stores [Cac2011]. According to [Bes2011], the dependability of Cloud storage can be realized by coupling and executing fault tolerance protocols such as Byzantine Fault Tolerance, and secret sharing protocols not in a single cloud but in the InterCloud environment.

In this chapter, I will present C-SCIT, an extension of SCIT that exploits the characteristics of the InterCloud in order to provide IT-QoS to services in SOA, as was first described in [Ng2011]. At the high level, C-SCIT is reusing the pattern of SCIT as a recovery-based intrusion tolerance architecture, with the addition of components to adapt to the multi-cloud environment.

9.2 C-SCIT Architecture

In addition to the components listed in the architecture to support services in an SOA environment, namely the Central Controller, Application Nodes, and Service Registry, C-SCIT architecture also comprises a Proxy and Local Controllers.

The main role of the *Central Controller* remains the same moving from SCIT to C-SCIT. In fact, it continues to control the cleansing procedure of the Application Nodes, but in an indirect fashion. Instead of sending signals to change the states to the Application Nodes, the Controller initiates the process by issuing operational messages to the SCIT Local Controllers, each of which resides on a virtual machine within its own Cloud Provider data center.

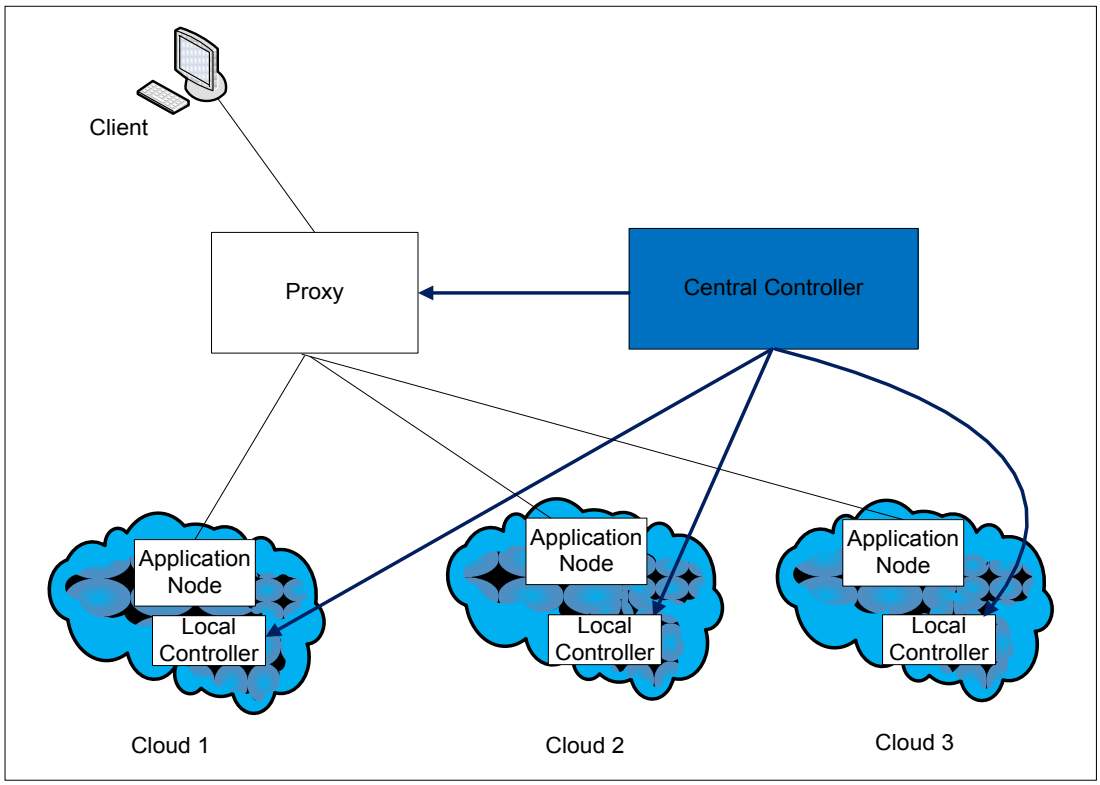

Figure 9.1. System Architecture of C-SCIT

Figure 9.1 [Ng2011] shows the Central Controller having communication links with three Local Controllers in Cloud 1, Cloud 2 and Cloud 3 respectively. The Central Controller itself is deployed in a Cloud, which should be separate from the Clouds where the Application Nodes are running. Another responsibility performed by the Central Controller is to ensure that the virtual images of the services are updated, patched, and devoid of malware, since they will be used for the continuous cleansing process. Moreover, the Central Controller interacts with the Service Registry containing information about the managed services, including their metadata, IT-QoS and SCIT operational data, images, and checksums. For managing the Application Nodes via the Local Controllers, the Central Controller invokes the Algorithm Modules that compute SCIT operational parameters based on the desired levels of intrusion tolerance.

As in the case of SCIT, Services to be protected by the C-SCIT will be loaded in the containers whose execution platform is virtualized on an *Application Node*. Thus, the replicas of a particular Service will run on different Application Nodes. The number of required replicas depends on the operating exposure window, which represents the level of intrusion tolerance, as shown in chapter 3. Each separate Cloud data center can run a Node, and store a copy of the service image. A copy of the service image is stored locally "in proximity" with the Node to make the cleansing quick and efficient. However, some procedures need to be in place to ensure that the service image is appropriately updated and integral.

In each Cloud data center, there is a *SCIT Local Controller* responsible of directly interacting with the Nodes in the same Cloud by relaying signals from the Central

Controller for the self-cleansing procedure. The introduction of Local Controller is to minimize the control traffic over the WAN from the Central Controller. Indeed, as I will show later, for cleansing, and activating Application Nodes, the detailed control steps will be initiated by the Local Controllers.

Proxy is the front-end component that is exposed to clients to mediate clients' requests and services' responses, because the Proxy is up to date about which Application Node is active at a specific time. For this, the Proxy regularly receives instructions from the Central Controller at every activation and cleansing cycle.

9.3 Discussion

By adapting the SCIT architecture pattern to the Cloud environments, C-SCIT has the potential of exploiting the benefits of Cloud Computing, while having to resolve challenges that stem from the new paradigm. The benefits are:

Distributed System. The InterCloud brings the distribution of the system and service to another level. Since C-SCIT operates in the InterCloud, the Application Nodes are deployed among different Cloud service providers, with data centers most likely located in various geographical sites. Thus, by nature, C-SCIT enhances the availability and reliability of services against natural disaster. Although Cloud providers have offered the option for application service delivery from different geographical sites, the InterCloud model that C-SCIT is based on is more guaranteed.

Diversity. Since each Application Node is running on the specific platform of a Cloud vendor, diversity, which is one of the factor for intrusion tolerance, comes naturally. The diversity present among Cloud providers permeates through all layers of the architecture,

from hardware servers, network devices, OS, hypervisors, and COTS products. As shown in Chapter 7, this diversity decreases the service attack surface, and improves its resilience.

Elastic Resources. Cloud Computing is known to offer elastic computing resources thanks to multi-tenancy and virtualization. On the economic side, the "pay-as-you-go" model provides a cost effective that services can use to adapt to changing load. Applying to the case of C-SCIT, this computing elasticity allows SCIT to rapidly adapt to changing environments in terms of intrusion tolerance. More precisely, the dynamic algorithm described in Chapter 8 will be simplified, since the resource constraint (in terms of maximum number of virtual nodes) can be lifted to a large extent.

Along with the advantages, C-SCIT has to consider issues related to connection dependability, as C-SCIT operates in the InterCloud environment.

From the system architecture of C-SCIT, the Central Controller has to communicate with the SCIT Local Controllers spread over multiple clouds. Since this connection is realized over the WAN, it is vulnerable to both standard reliability and S-Reliability.

- First, if the Central Controller cannot send SCIT control signals to the Local Controllers due to link failure, the cleansing process may be disrupted. Consequently, requirements of IT-QoS such as MTTSF, S-Availability and S-Reliability of the services protected by C-SCIT may not be satisfied. In order to overcome this problem, the implementation of C-SCIT must rely on a pre-established and adequate SLA (Service Level Agreement) in terms of communication links within the InterCloud.

- Moreover, the SLA should include the reliability in terms of security. Recall that the SCIT architecture pre-supposes that the communication between the Central Controller and the Application Nodes is trusted, so that the Central Controller cannot be compromised. This stringent condition can be realized when the Controller and the Nodes are deployed in a highly secure enclave within an enterprise network [Hua2006]. Trusted wormhole as described in MAFTIA [Str2004] or some hardware solution [Hua2007] can be used to guarantee this secure link requirement. In C-SCIT, the connection between an Application Node and its managing Local Controller can follow the options just mentioned. As for the communication from the Central to Local Controllers, dedicated VPN (Virtual Private Network) links represent an option to ensure the secure transmission void of possible tampering and spoofing of the SCIT control signals.
- Each service will have a virtual image. The Central Controller will have a management scheme to ensure that such virtual images used for the cleansing are uncorrupted and up-to-date with patches. On one hand, this scheme will be reflected in the steps within the activation procedure laid out in a later section below. On the other hand, a special Service Registry (???), to be accessed exclusively by the Central Controller, can be used as a repository for the services' images and their associated checksums.

9.4 Self-Cleansing Procedure

SCIT self-cleansing consists of two sub-procedures: cleansing of the service instance currently online, and activation of live spare instance of the same service. In the

discussion below, I assume that the service registry contains the authoritative images used to load the services, and the links between the Central and Local Controllers are secure.

9.4.1 Cleansing Procedure

Figure 9.2 [Ng2011] depicts the collaboration of the main components during the cleansing procedure of a service residing in an Application Node. Co-located in the same cloud as this Application Node is the Local Controller. The cleansing steps are as followed:

- Step 1. The Central Controller queries the Local Controller to get the checksum of the service image, using the operation queryImageChecksum().
- Step 2. The Central Controller then get the checksum from the Service Registry using getChecksum(), and verifies whether the checksum provided in the Local Controller's response matches the one in the Service Registry. If there is an exact match, meaning that the image stored in the Cloud is up-to-date and uncorrupted, then the procedure proceeds to Step 3.1. Otherwise, go to Step 3.2.
    - Step 3.1. The Central Controller the Local Controller to use the service image currently in the Local store by invoking authorizeLoadImage().
    - Step 3.2. The Central Controller transfers the new image to the Local Controller, which deposits in its Local store by invoking transferImage().
- Step 4. Now that the Local store has the most current and pristine service image. The Local Controller loads that image to the allocated Application Node using loadImage().

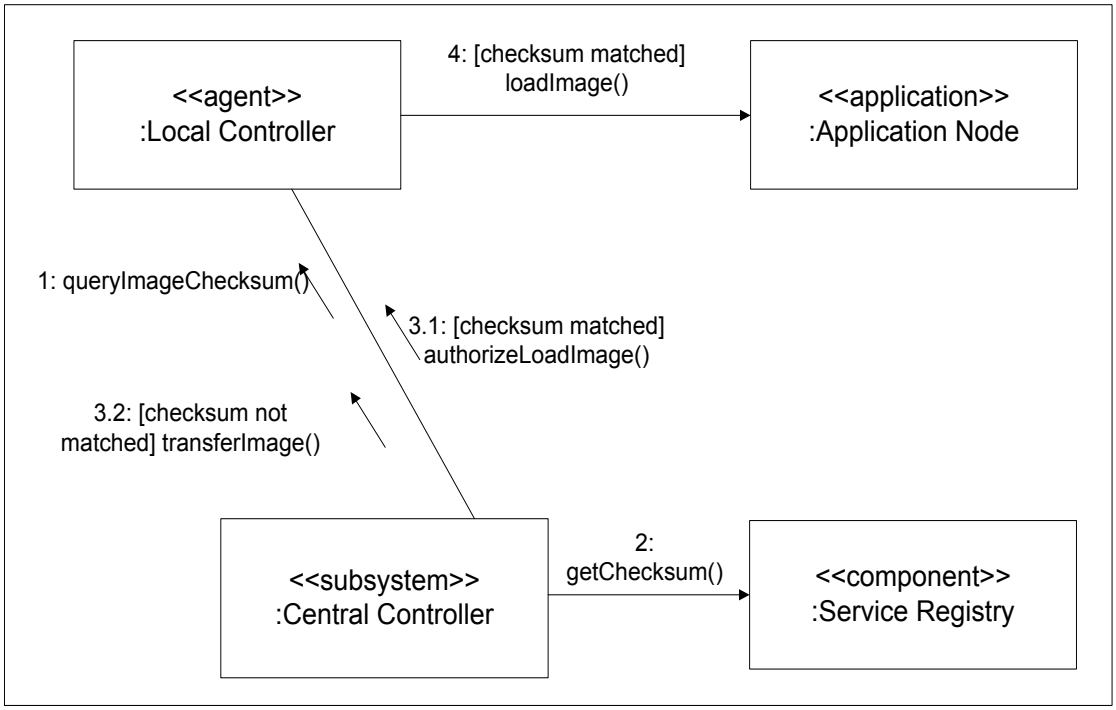

Figure 9.2. C-SCIT Cleansing Procedure.

9.4.2 Activation Procedure

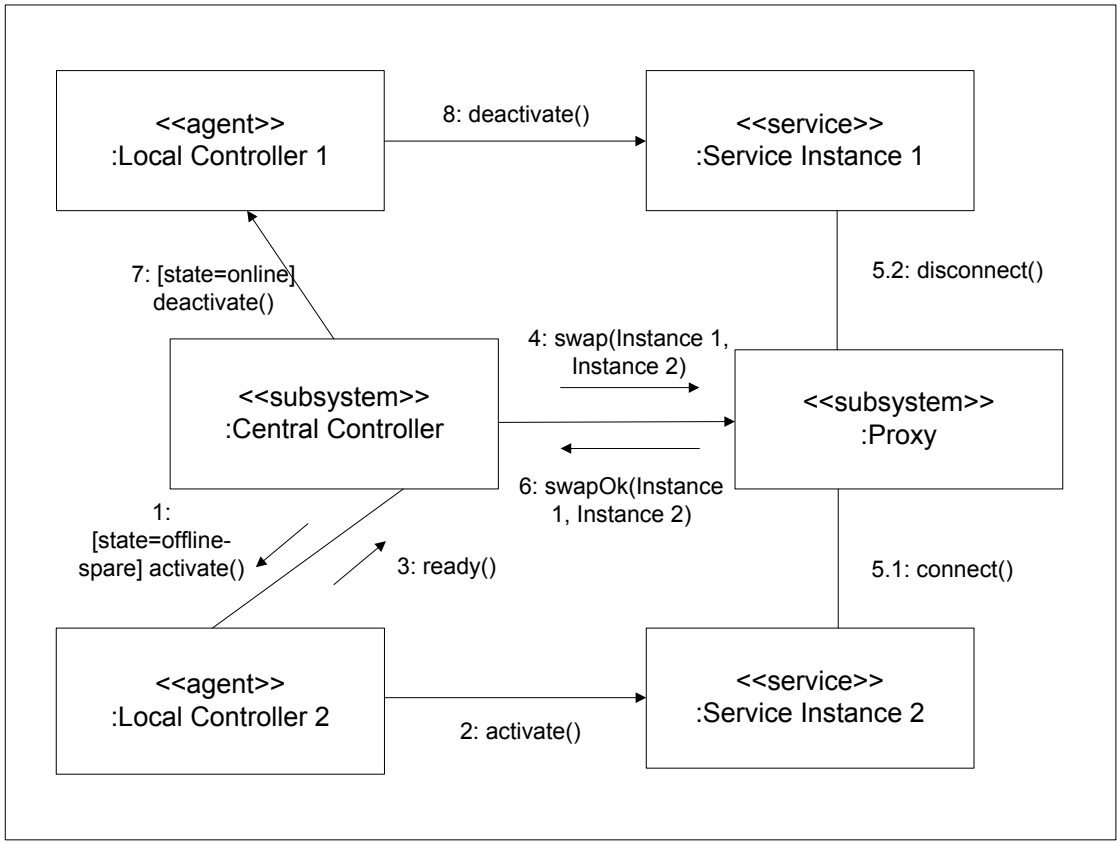

Figure 9.3. C-SCIT Activation Procedure Diagram.

Figure 9.3 [Ng2011] depicts the collaboration of the main components during the activation procedure, which involves deactivating the online service instance (Service Instance 1) managed by Local Controller 1, and activating the offline-spare service instance (Service Instance 2) managed by Local Controller 2. The activation steps are as followed:

- Step 1. The Central Controller activates the offline spare Service Instance by sending the activate() signal to the Local Controller (number 2 in the diagram), which perform the actual activation of the service instance (number 2) being in offline spare state.

- Step 2. After receiving the success message from Local Controller 2, the Central Controller proceeds to swap the service instances in order to make Service Instance 2 the online instance to receive transaction requests. The swapping is realized by changing the configuration of the proxy via the swap() operation. This translates to connecting Service Instance 2 to the Proxy and disconnecting Service Instance 1 from the Proxy at the same time.
- Step 3. Upon successful swapping operation, the Central Controller requests Local Controller 1 to deactivate Service Instance 1.

9.5 Impact on SCIT Parameters

SCIT operation parameters include the cleansing time T_C, the exposure window W_O, the grace period W_{GP}, and the number of nodes N. With C-SCIT, network latency has to be taken into account since the Central Controller and Local Controllers can reside in different clouds.

Let $C = \{C_1, \ldots, C_p\}$ be the set of available Clouds.

Definition 9.1. Within the usage of C-SCIT, a Cloud C_i is determined by the following tuple:

$C_i = (t_{i,load}, t_{i,com}, t_{i,verify}, t_{i,correct}, P_{i,cf})$,

where $t_{i,load}$ is the image loading time, $t_{i,com}$ the communication latency, $t_{i,verify}$ the checksum verification time, $t_{i,correct}$ correction time, and $P_{i,cf}$ the probability that checksum test fails, and maybe.

Proposition 9.1.

The average cleansing time for a service in C-SCIT is given by:

$$t_{i,C} = t_{i,load} + t_{i,com} + t_{i,verify} + P_{i,cf} * t_{i,correct} \quad (9.1).$$

The overall cleansing time for C-SCIT system is given by:

$$T_C = \max_{i=1,..,p}\{t_{i,C}\} \quad (9.2).$$

The overall activation time will be the maximum of all activation times:

$$T_A = \max_{i=1,..,p}\{t_{i,A}\} \quad (9.3).$$

Proposition 9.2. The number of pairs (Local Controller, Service Instance) required to achieve the given exposure window WO can be estimated by:

$$N = \left\lceil \frac{T_C + T_A}{W_o} \right\rceil + 2 \quad (9.4).$$

Proof.

I have to add TA to the formula (3.8) in chapter 3:

$$N = \left\lceil \frac{W_{GP} + T_C + T_A}{W_o} \right\rceil + 1.$$

If I let $W_{GP} = W_o$, then the above equality becomes:

$$N = \left\lceil \frac{T_C + T_A}{W_o} \right\rceil + 2.$$

Example. If T_C is equal to 12 min and T_A = 2 min, and W_o = 3 min, then N = 7 pairs, according to (9.4).

9.6 Providing IT-QoS Levels

In this section, the algorithms established in the previous chapter to provide IT-QoS levels will be revised to adapt to C-SCIT. At the high level, the new algorithms will reuse the modules to compute the exposure windows to satisfy the different IT-QoS levels as

described in Chapter 8. At the same time, they will have to: a) implement the steps described in the previous subsections for cleansing and activation procedures; b) modify the calculation of the required number of nodes based on the new formula (9.4). At the core is the function calculateConfig() which computes the exposure window and number of nodes, given the value of an IT-QoS parameter. The Central Controller will execute the algorithm listed in Table 9.1 to perform periodic cleansing in the InterCloud while ensuring the given IT-QoS parameter, based on the analytical results obtained in Chapter 4.

Table 9.1. C-SCIT Operation Algorithm.

C-SCIT-Operate()
Input • q: IT-QoS object, including type t and value v. • $Cl=\{Cl_1,..,Cl_p\}$: set of available Clouds; • λ_A, λ_F, λ_2.
1. $(W, N) \leftarrow$ calculateConfig(q, Cl); 2. Build queue Q_g with first N Clouds taken from Cl; 3. Fork a thread and invoke cleanse(Q_g); 4. Fork a thread and invoke activate(Q_g); **calculateConfig(q, Cl)** 5. Compute T_C using (9.2). 6. Compute T_A using (9.3). 7. switch (q.t) 8. case MTTSF: 9. Call configureMTTSF() given in Table 8.1. 10. case SA: 11. Call computeSA() given in Table 8.2 12. Let W_{sol} be the solution. 13. $W_o \leftarrow W_{sol}$; 14. $W_{GP} \leftarrow W_{sol}$; 15. Compute N using (7);

```
16. return (W_o, N);

cleanse(Q_g)
17. While TRUE
18.   Cl_i ← dequeue(Q_g);
19.   Perform cleansing steps of Cl_i as in section
      9.4.1.
20.   enqueue(Q_s, Cl_i);
21.   Wait(timer1);
22. EndWhile

activate(Q_g)
23. While TRUE
24.   Cl_a ← currently active Cloud;
25.   Cl_s ← dequeue(Q_s);
26.   If Cl_s = null
27.     return error;
28.   EndIf
29.   If Cl_a = null
30.   Then
31.     Activate Cl_s;
32.   Else
33.     Perform activation steps as in section 9.4.2.
      This includes Cl_a and Cl_s;
34.   EndIf
35.   Wait(timer2);
36. EndWhile
```

The algorithm makes use of two queues:

- Queue Q_g built out of a subset of Cl. At one point in time, only the Clouds in Q_g are utilized by the Central Controller.
- Queue Q_s contains the Clouds, each with a pair of Local Controller and Service Instance.

C-SCIT-Operate contains the following steps.

Step 1. Based on the value of the IT-QoS, calculate the exposure window and the number of nodes N.

Step 2. Reserve N Clouds and enqueue them in Q_g.

Step 3. Execute the cleansing and activation procedures described in a previous section, based on the exposure window, the cleansing time and activation time.

The dynamic reconfiguration of the C-SCIT is much simplified, as compared to the case of SCIT deployed in the enterprise, thanks to the elastic provision of the number of nodes, and computing resources. In fact, the dynamic scheme will consists of two steps:

- Upon receiving the alarm from the monitor, kill the current threads executing the cleansing and activation.
- Call C-SCIT-Operate() by passing the new value for the attack rate.

9.7 Cloud APIs

The implementation of the algorithm C-SCIT-Operate() can be realized by the APIs provided by Cloud service providers such as Amazon Web Services (AWS) [Ama2014] and Windows Azure [Win2014]. The main functionalities needed from the Cloud providers are used for:

- activating a service instance via the web service AWS *StartInstances* web service and Microsoft's *CreateHostedService*.
- deactivating via the command *ec2-terminate-instances* or the RESTFul web service *ec2-stop-instances* for AWS and the *DeleteHostedService* for Windows Azure.
- cleansing via the APIs

- storing the image of the service instance.

9.8 Summary

In this chapter, I have presented C-SCIT, an extension of SCIT to operate in the InterCloud that leverages the computing capabilities of the Cloud paradigm. C-SCIT reuses the analytical results obtained in chapter 4, and comprises steps to control the cleansing over multiple clouds. I also have shown the possibility of the implementation thanks to the available web services provided by the commercial Cloud vendors.

CHAPTER 10 - CONCLUSION

10.1 Summary

In this research, I have established the QFITS framework, that allows system and software architects to design services in an SOA environment in such a way that they satisfy desired levels of IT-QoS. The design is backed by a quantitative analysis based on Semi-Markov Process. Moreover, with SCIT as the foundational mechanism for intrusion tolerance, QFITS enables an end-to-end engineering to achieve desired IT-QoS.

1) QFITS provides a design framework based on SCIT pattern. Thanks to the pattern, the security problem is transposed to a domain in which well-known mathematical tools can be employed, as for fault-tolerant systems and services. In the dissertation, I have demonstrated that Semi-Markov Process can be used to model SCIT-based services, so that from the solution, a correlation between the intrusion tolerance control parameters and the IT-QoS characteristics can be derived.

2) If SCIT is combined with other intrusion tolerance mechanisms, such as intrusion detection, the quantitative methods have been shown feasible. The exposure window, SCIT's main control parameter, offers an abstraction that facilitates the quantitative methods based on Semi-Markov Process and correlation computations.

3) Since SCIT is a recovery-based intrusion tolerance with periodic cleansing, it is also viewed as time-based, and represents a complement to the space-based strategy in order to reduce the service's attack surface.

4) The mathematical expressions obtained for the IT-QoS parameters allow the systematic design of services with certain IT-QoS guarantees. Within the SOA paradigm, the concept of SCIT-based enabler service will serve as a foundation for designing services in the SOA architecture so that these services also have IT-QoS guarantees.

10.2 Future Work

There are outstanding issues that can constitute directions for future work. In this research, I have demonstrated the feasibility of QFITS with simple SCIT, SCIT combined with IDS, and SCIT with Diversity. As shown by the taxonomy of intrusion tolerant architectures, mechanisms other than SCIT do exist. Therefore, the question is whether to make QFITS work when those intrusion tolerance schemes are employed.

In the area of Semi-Markov process analysis, it would be useful to investigate the possibility of composing Semi-Markov processes. If the composition could be done, then one could analyze separate small Semi-Markov processes, then use the composition theory and rules to combine processes. By decomposing a problem into smaller ones, we can avoid the complexity of the analysis; simultaneously, the methodology lends itself naturally for SOA, where composing services is the norm to build more complex services and applications.

Regarding the SOA paradigm, this research presents the perspectives of a service provider designing services with IT-QoS levels. However, it would be useful in a future work to adapt QFITS to handle web services provided by different service providers.

Finally, QFITS is applicable at the service level. The question is whether it is possible to extend SCIT to components that are more granular than a service. For instance, can the recovery-based intrusion tolerance scheme be applied to modules, functions, and objects of the service?

APPENDIX A - SIMPLE SCIT SIMULATION CODE

A.1 Main Class - SmpSim

```
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Date;
import java.util.Scanner;

import org.apache.commons.math3.distribution.ExponentialDistribution;
import org.apache.commons.math3.distribution.UniformRealDistribution;

public class SmpSim {
	// SMP states
	int Good = 1;
	int Attack = 2;
	int Fail = 3;
	int currentState = 1;
	int nextState = 0;

	// Member fields
	double[] Wvalues = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
	double exposureW;
	double activeW;
	UniformRealDistribution cleansingTimeatG;
	UniformRealDistribution cleansingTimeatA;
	ExponentialDistribution arrivalTime;
	ExponentialDistribution compromiseTime;
	double arrivalMean = 1.0;
	double compromiseMean = 1.0;

	// Data collectors
	int iterationNum = 10;  // number of iterations
	int runNum = 10000;     // number of runs to compute the mean
	//double[] mttsfArray = new double[iterationNum];
	WindowQoS[] totalResult;

	// I/O tool
	FileWriter fw;
	FileWriter fw2;
	BufferedWriter bw;
	BufferedWriter bw2;


	// Default Constructor
```

```
	public SmpSim() {
	}

	// Configure for simulation
	public void init(String[] args) {

		// distributions
		arrivalTime = new ExponentialDistribution(arrivalMean);
		compromiseTime = new
ExponentialDistribution(compromiseMean);

		try {
		   String filename = "sim2_am=" + arrivalMean + "_cm=" +
compromiseMean + ".txt";
		   File file = new File(filename);

		   String filename2 = "sim2b_am=" + arrivalMean + "_cm=" +
compromiseMean + ".txt";
		   File file2 = new File(filename2);

		   if (!file.exists()) {
		      file.createNewFile();
		   }
		      if (!file2.exists()) {
		            file2.createNewFile();
		      }

		      fw = new FileWriter(file.getAbsoluteFile());
		      fw2 = new FileWriter(file2.getAbsoluteFile());

		} catch (IOException e) {
		      e.printStackTrace();
		}

		// Data Collector
		totalResult = new WindowQoS[Wvalues.length];

	}

	// Run simulation until state F is reached
	// Input: active window aW; exposure window eW
	// Return ttsf: Time to Security Failure
	public double runUntilFail(double aW, double eW, double[] av) {
		currentState = Good;
		double ttsf = 0.0;
		double at = 0.0;
		double ct = 0.0;
		double cleanTimeatG = 0.0;
		double cleanTimeatA = 0.0;
		double sojournTimeF = 0.0;

		while (currentState != Fail) {
		      if (currentState == Good) {
```

```
                        at = arrivalTime.sample();
                        cleanTimeatG = cleansingTimeatG.sample();
                        if (at < cleanTimeatG) {
                              // go to state A
                              currentState = Attack;
                              ttsf = ttsf + at;
                        }
                        else {
                              ttsf = ttsf + cleanTimeatG;
                        }
                  }
                  else if (currentState == Attack) {
                        ct = compromiseTime.sample();
                        cleanTimeatA = cleansingTimeatA.sample();
                        if (ct < cleanTimeatA) {
                              // go to state F
                              currentState = Fail;
                              ttsf = ttsf + ct;
                        }
                        else {
                              currentState = Good;
                              ttsf = ttsf + cleanTimeatA;
                        }
                  }
                  if (currentState == Fail) {
                        sojournTimeF = cleansingTimeatA.sample();
                        av[0] = sojournTimeF;
                  }
            }
            return ttsf;
      }

      // Run simulation
      // aW: active window
      // eW: exposure window
      public void run(double aW, double eW, double[] mttsfArray,
double[] AvArray) {
            double mttsf = 0.0;
            double mttsr = 0.0;
            double[] ttsr = {0.0};

            // Get iterationNum values for MTTSF
            for (int i=0; i<mttsfArray.length; i++) {

                  // Run the SMP runNum times to get the average
                  for (int j=0; j<runNum; j++) {
                        double ttsf = runUntilFail(aW, eW, ttsr);
                        mttsf = mttsf + ttsf;
                        mttsr = mttsr + ttsr[0];
                  }
                  mttsf = mttsf/runNum;
                  mttsr = mttsr/runNum;
                  mttsfArray[i] = mttsf;
```

```
                AvArray[i] = mttsf / (mttsf + mttsr);
            }
        }

        // Run simulation for different values of W
        public void totalRun() {
            for (int i=0; i<Wvalues.length; i++) {
                double aW = Wvalues[i]/2;
                double eW = Wvalues[i];
                totalResult[i] = new WindowQoS(eW, iterationNum);
                cleansingTimeatG = new UniformRealDistribution(0.0,
aW);
                cleansingTimeatA = new UniformRealDistribution(0.0,
eW);
                run(aW, eW, totalResult[i].QoS, totalResult[i].Av);
            }
        }

        // Output results to a file
        public void output() {
            try {
                // initialize output file
                bw = new BufferedWriter(fw);
                bw2 = new BufferedWriter(fw2);
                Date d = new Date();
                bw.write("Class="+ this.getClass().getName() + ";
Date=" + d); bw.newLine();
                bw.write("arrivalMean=" + arrivalMean + ";
compromiseMean=" + compromiseMean);
                bw.newLine(); bw.newLine(); bw.newLine();

                for (int i=0; i<totalResult.length; i++) {
                    for (int j=0; j<totalResult[i].QoS.length; j++)
{
                        bw.write(totalResult[i].W + ", " +
totalResult[i].QoS[j]); bw.newLine();
                        bw2.write(totalResult[i].W + ", " +
totalResult[i].Av[j]); bw2.newLine();
                    }
                }
                bw.flush(); bw.close(); bw2.flush(); bw2.close();
            }
            catch (IOException e) {
                e.printStackTrace();
            }

        }

        public void getInput() {
            String s;
            Scanner in = new Scanner(System.in);
            System.out.println("ArrivalTime Mean: ");
            s = in.nextLine();
```

```
            arrivalMean = Double.parseDouble(s);
            System.out.println("You entered Arrival Mean: "+
arrivalMean);

              System.out.println("CompromiseTime Mean: ");
              s = in.nextLine();
            compromiseMean = Double.parseDouble(s);
            System.out.println("You entered Compromise Mean: "+
compromiseMean);
      }

      public static void main(String[] args) throws Exception {
            System.out.println("Start SMP Simulator");
            SmpSim sim = new SmpSim();
            sim.getInput();
            sim.init(args);
            sim.totalRun();
            sim.output();
            System.out.println("End SMP Simulator");
      }
}
```

A.2 Auxiliary Class - WindowQoS

```
public class WindowQoS {
      public double W;
      public double[] QoS;  // MTTSF
      public double[] Av;   // S-Availability

      WindowQoS (double w, int num) {
            W = w;
            QoS = new double[num];
            Av = new double[num];
      }
}
```

REFERENCES

[Ama2014] Amazon Elastic Compute Cloud (EC2). http://aws.amazon.com/ec2/.

[Arn2014] Arnold, Florian, Holger Hermanns, Reza Pulungan, and Marielle Stoelinga. "Time-dependent Analysis of Attacks." [private Communication].

[Aun2005] Aung, Khin Mi Mi, Kiejin Park, and Jong Sou Park. "A Rejuvenation Methodology of Cluster Recovery". *CCGrid 2005, IEEE International Symposium* Vol. 1, Pp. 90 - 95, May 2005.

[Axe2000] Axelsson, Stefan. "The Base-rate Fallacy and the Difficulty of Intrusion Detection." *ACM Transactions on Information and System Security* 3.3 (2000): 186-205.

[Ban2009] Bangalore, Anantha K., and Sood, Arun K.. "Securing Web Servers Using Self Cleansing Intrusion Tolerance (SCIT)." *DEPEND '09. Second International Conference on Dependability, 2009.* Proc. of Second International Conference on Dependability, Pp. 60-65, Athens, 2009.

[Bes2011] Bessani, Alysson, Miguel Correia, Bruno Quaresma, Fernando Andre, and Paulo Sousa. "DepSky: Dependable and Secure Storage in a Cloud-of-Clouds". EuroSys'11, April 10-13, 2011, Salzburg, Austria. http:// www.di.fc.ul.pt/~mpc/pubs/eurosys219-bessani.pdf. *[02/28/2011]*.

[Boc2008] Bocciarelli, Paolo, and Andrea D'Ambrogio. "Model-driven Performability of Composite Services". *SIPEW 2008*, LNCS 5119, Pp. 228-246, 2008.

[Buy2008] Buyya, R., Yeo Chee, and S. Venugopal. "Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities". 10th International Conference on High Performance Computing and Communications, 2008. HPCC '08. Sep. 25-27, 2008.

[Buy2010] Buyya, Rajkumar, Ranjan, and Rodrigo N. Calheiros." InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services". ICA3PP 2010, Part I, LNCS 6081, pp. 13–31, 2010.

[Cac2011] Cachin, Christian, Robert Haas, and Marko Vukolic. “Dependable Storage in the Intercloud”. http://domino.research.ibm.com/library/cyberdig.nsf/papers/630549C46339936C852577C200291E78. [02/27/2011].

[Car2009] Cardellini, Valeria, Emiliano Casalicchio, Vincenzo Grassi, Francesco Lo Presti, and Raffaela Mirandola. “QoS-driven Runtime Adaptation of Service Oriented Architectures”. ESEC-FSE’09, Amsterdam, The Netherlands, August 2009.

[Clo2014] CloudSim: A Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services. Http://www.cloudbus.org/cloudsim/ [20 Jan. 2014].

[Con2002] The Consultative Committee for Space Data Systems. “Reference Model for an Open Archival Information System (OAIS)”, 2002. Available: Http:// Public.ccsds.org/publications/archive/650x0b1.pdf [Feb. 25, 2009].

[Cuk2001] Cukier, Michel et al. "Intrusion Tolerance Approaches in ITUA". *Supplement of the 2001 International Conference on Dependable Systems and Networks*, Göteborg, Sweden, July 1-4, 2001, pp. B-64 to B-65.

[Dam2006] D’Ambrogio, Andrea. "A Model-driven WSDL Extension for Describing the QoS of Web Services." Proc. of IEEE International Conference on Web Services (ICWS’06), 2006, USA, Chicago.

[Gaf2001] Gaffney, John E. Jr. Ulvila, Jacob W. “Evaluation of Intrusion Detectors: A Decision Theory Approach”. IEEE Symposium on Security and Privacy, 2001, pp. 50-61.

[Gan003] Ganger, Gregory R. ; Khosla, Pradeep K. "PASIS: A Distributed Framework for Perpetually Available and Secure Information Systems". http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA436245. [Jan 31, 2014].

[Heu2010] Heumann, Thomas, Sven Turpe, and Jorg Keller. "Quantifying the Attack Surface of a Web Application". Sicherheit, Volume 170 of LNI, pp. 305-316. GI, 2010.

[How2002] Howard, Michael, John Pincus, and Jeannette M. Wing. “Measuring Relative Attack Surfaces”. Http://www.cs.cmu.edu/~wing/publications/Howard-Wing03.pdf.

[Hua2011] Huang, Yih, and Anup K. Ghosh. “Introducing Diversity and Uncertainty to Create Moving Attack Surfaces for Web Services”. Advances in Information Security, 1, Vol. 54, Moving Target Defense, Pp. 131-151. 2011.

[Hua2006] Huang, Yih, David Arsenault, and Arun Sood. “Incorruptible System Self-Cleansing for Intrusion Tolerance”, *Performance, Computing, and Communications Conference, IPCCC 2006*.

[Hua2007] Huang, Yih, David Arsenault, and Arun Sood. “Secure, Resilient Computing Clusters: Self-Cleansing Intrusion Tolerance with Hardware Enforced Security (SCIT/HES)”, *The Second International Conference on Availability, Reliability, and Security, ARES 2007*.

[Hua2006a] Huang, Yih, David Arsenault, and Arun Sood. “Securing DNS Services through System Self Cleansing and Hardware Enhancements”, *The First International Conference on Availability, Reliability, and Security, ARES 2006*.

[Hua2003] Huang, Yih, Ravi Bhaskar, and Arun Sood. “Countering Web Defacing Attacks with System Self Cleansing”, *Proceedings of 7th World Multiconference on Systemics, Cybernetics and Informatics*, Orlando, Florida, July 2003, Pp. 12-16.

[Irv2001] Irvine, Cynthia, and Timothy Levin. “Quality of Security Service”. *Proceedings of the 2000 Workshop on New Security Paradigms*, Pp. 91-99, Ireland, 2001.

[Jan2006] Janssen, Jacques, and Raimondo Manca. *Applied Semi-Markov Processes*. New York, NY: Springer Science Business Media, 2006.

[Jus2002] Just, J. E., J. C. Reynolds, and K. Levitt, “Intrusion tolerance through forensics-based attack learning,” in Intrusion Tolerant System Workshop, Supplemental Volume on 2002 International Conference on DependablenSystem and Networks, pp. C–4–1, 2002.

[Kar1972] Karp, Richard M. In *Complexity of Computer Computations* (eds. R. E. Miller, and J. W. Thatcher), Pp. 85-104, New York: Plenum Press, 1972.

[Kea2009] Keahey, Katarzyna, Mauricio Tsugawa, Andrea Matsunaga, Jose Fortes, "Sky Computing," *IEEE Internet Computing*, vol. 13, no. 5, pp. 43-51, Sept.-Oct. 2009, doi:10.1109/MIC.2009.94.

[Kni2002] Knight, J., D. Heimbigner, and A. Wolf. “The Willow Architecture: Comprehensive Survivability for Large-Scale Distributed Applications”, *lntrusion Tolerance System Workshop, Supplemental Volume On 2002 International Conference on Dependable .System and Network*, 2002.

[Kob2012] Kobayashi, Hisashi, Brian L. Mark and William Turin. *Probability, Random Processes, and Statistical Analysis: Applications to Communications, Signal Processing,*

Queueing Theory and Mathematical Finance. Cambridge University Press (February 13, 2012).

[Kre2010] Kreidl, Patrick O. "Analysis of a Markov Decision Process Model for Intrusion Tolerance". Proceedings of the 2010 International Conference on Dependable Systems and Networks Workshops (DSN-W), pp. 156-161.
[Lil2000] Lilja, David J. *Measuring Computer Performance: A Practitioner's Guide*. Cambridge, UK: Cambridge UP, 2000.

[Lim2001] Limnios, N., and Gheorghe Oprişan. *Semi-Markov Processes and Reliability*. Boston: Birkhäuser, 2001. ISBN 0-8176-4196-3.

[Liu2003] Liu, Peng. *Architectures for Intrusion Tolerant Database Systems*. Proceedings of the Foundations of Intrusion Tolerant Systems (OASIS '03), 2003.

[Liu2008] Liu, Yanguo. "Quantitative Security Analysis of Service-Oriented Software Architectures". PhD Thesis, 2008. http://dspace.library.uvic.ca:8080/bitstream/handle/1828/895/PhD_Dissertation-Michael_Liu-Final.pdf?sequence=1.

[Ma2004] Madan, B. "A Method for Modeling and Quantifying the Security Attributes of Intrusion Tolerant Systems." *Performance Evaluation* 56.1-4 (2004): 167-86.

[Man2011] Manadhata, Pratyusa K., and Jeannette M. Wing. "An Attack Surface Metric." *IEEE Transactions on Software Engineering* 37.3 (2011): 371-86.

[Men2002] Menascé, Daniel A., and Virgilio A. F. Almeida. *Capacity Planning for Web Services: Metrics, Models, and Methods*. Upper Saddle River, NJ: Prentice Hall, 2002.

[Men2010] Menasce, Daniel A., João P. Sousa, Sam Malek, and Hassan Gomaa. "QoS Architectural Patterns for Self Architecting Systems." Proc. of 7th International Conference on Autonomic Computing, USA, New York, 2010, Pp. 195-204.

[Nag2010] Nagarajan, Ajay, and Arun Sood. "SCIT and IDS Architectures for Reduced Data Ex-filtration" 4th Workshop on Recent Advances in Intrusion-Tolerant Systems, Chicago,IL, USA, June 28 2010.

[Nag2011] Nagarajan, Ajay , Quyen Nguyen, Robert Banks and Arun Sood, "Combining Intrusion Detection and Recovery for Enhancing System Dependability", 5th Workshop on Recent Advances in Intrusion-Tolerant Systems, in conjunction with 41st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2011), Hong Kong, 28 June, 2011.

[Ng2009] Nguyen, Quyen, and Arun Sood. “Quantitative Approach to Tuning of a Time-Based Intrusion-Tolerant System Architecture”. WRAITS 2009, Lisbon, Portugal.

[Ng2010] Nguyen, Quyen, and Arun Sood. “Comparative Analysis of Intrusion-Tolerant System Architecture”. " *IEEE Security and Privacy*, 30 Aug. 2010. IEEE Computer Society Digital Library.
[Ng2010a] Nguyen, Quyen L, and Arun Sood, "Multiclass S-Reliability for Services in SOA", accepted for The Fifth International Conference on Software Engineering Advances, ICSEA 2010, Nice, France, August 22-27, 2010.

[Ng2011] Nguyen, Quyen, and Arun Sood, “Designing SCIT Architecture Pattern in a Cloud-based Environment”, The First International Workshop on Dependability of Clouds, Data Centers and Virtual Computing Environments (DCDV 2011) in conjunction with 41st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2011), Hong Kong, 28 June, 2011.

[Ng2012] Nguyen, Quyen L., and Arun K. Sood. "Improving Resilience of SOA Services along Space-time Dimensions." *Dependable Systems and Networks Workshops (DSN-W), 2012 IEEE/IFIP 42nd International Conference on* 1.6 (2012): 25-28.

[Ng2013] Nguyen, Quyen L., and Arun K. Sood. “Building a Resilient Service-Oriented Architecture Environment”, CrossTalk, September/October 2013.

[Oas2004] OASIS. UDDI Version 3.0.2. Available: Http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm.

[Obj2008] Object Management Group. “UML Profile for Modeling Quality of Service & Fault Tolerance Characteristics & Mechanisms, V1.1”, April 2008.

[Obr2003] O'Brien, D. ” Intrusion tolerant Web servers via network layer controls". DARPA Information Survivability Conference and Exposition, 2003. Proceedings (Volume:2). 22-24 April 2003.

[Pal2009] Pal, Partha Et Al. “What Next in Intrusion Tolerance”. WRAITS 2009, Lisbon, Portugal. [Jan 31, 2014]: Http://wraits09.di.fc.ul.pt/wraits09paper1.pdf.

[Pal2007] Pal, Partha, Franklin Webber, and Richard Schantz. “The DPASA Survivable JBI – A High-Water Mark in Intrusion-Tolerant Systems”, *Workshop on Recent Advances in Intrusion Tolerant Systems’07*, 2007.

[Pha2006] Pham, Hoang. *System Software Reliability*. Berlin: Springer, 2006. ISBN 978-1852339500.

[Rei2007] Reiser, Hans P. and Rudiger Kapitza. "Hypervisor-Based Efficient Proactive Recovery". SRDS '07. Proceedings of the 26th IEEE International Symposium on Reliable Distributed Systems, pp. 83-92.

[Ros2009] Ross, Sheldon M. *Introduction to Probability Models*. Academic Press; 10 edition (December 17, 2009).

[Sag2014] "Sage: Open Source Mathematics Software."<http://sagemath.org/>. [Jan. 20, 2014].

[Sal2008] Saltelli, A. *Global Sensitivity Analysis: The Primer*. Chichester, England: John Wiley, 2008.

[Sch2005] Schechter, S.e. "Toward Econometric Models of the Security Risk from Remote Attacks." *IEEE Security and Privacy Magazine* 3.1 (2005): 40-44.

[Smi2008] Smith, Matthew, Christian Schridde, and Bernd Freisleben. "Securing Stateful Grid Servers through Virtual Server Rotation", *HPDC'08,* June 23–27, 2008, Boston, Massachusetts, USA.

[Sou2008] Sousa, Paulo, Alysson Neves Bessani, and Rafael R. Obelheiro. "The FOREVER Service for Fault/Intrusion Removal". WRAITS 2008, Glasgow, Scotland. Proceedings of the 2nd Workshop on Recent Advances on Intrusiton-tolerant Systems.

[Sou2007] Sousa, Paulo Et Al. "Resilient Intrusion Tolerance through Proactive and Reactive Recovery". *13th IEEE International Symposium on Pacific Rim Dependable Computing*, 2007.

[Str2004] Stroud, Robert, Ian Welch, John Warne, and Peter Ryan. "A Qualitative Analysis of the Intrusion-tolerance Capabilities of the MAFTIA Architecture", *2004 International Conference on Dependable Systems and Networks*, Issue 28 June-1 July 2004, Pp. 453 – 461.

[Tri2002] Trivedi, Kishor Shridharbhai. *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*. New York: Wiley, 2002.

[Uml2005] UML Profile for Schedulability, Performance and Time Specification. http://www.omg.org/spec/SPTP/. [Jan 31, 2014].

[Val2002] Valdes, A., M. Almgren, S. Cheung, Y. Deswarte, B. Dutertre, J. Levy, H. Saidi, V. Stavridou, and T. E. Uribe. "An architecture for an adaptive intrusion tolerant server". Proc. Security Protocols Workshop, LNCS, Springer-Verlag, 2002.

[Ver2006] Verissimo, P.e., N.f. Neves, C. Cachin, J. Poritz, D. Powell, Y. Deswarte, R. Stroud, and I. Welch. "Intrusion-tolerant Middleware: The Road to Automatic Security." *IEEE Security & Privacy Magazine* 4.4 (2006): 54-62.

[Ver2013] Verizon. "The 2013 Data Breach InvestigationsÂ Report." *Verizon Enterprise Solutions*. [22 Jan. 22, 2014] <http://www.verizonenterprise.com/DBIR/2013/>.

[Ver2007] Verma, Naresh, Yih Huang, and Arun Sood. "Proactively Managing Security Risk",*Security Focus, Nov. 7, 2007*. Available: Http://www.securityfocus.com/infocus/1896/1 [Feb. 24, 2009].

[Wan2003] Wang, Feiyi. "Analysis of Techniques for Building Intrusion Tolerant Server Systems." *IEEE 2003 Military Communications Conference, 2003, MILCOM '03*. Proc. of Military Communications Conference. Vol. 2, 2003, Pp. 729-34.

[Wan2003a] Wang, Feiyi, Frank Jou, Fengmin Gong, C. Sargor, K. Goseva-Popstojanova, and K. Trivedi. "SITAR: a Scalable Intrusion-tolerant Architecture for Distributed Services", *Proceedings of The Foundations of Intrusion Tolerant Systems (OASIS '03)*. Los Alamitos, CA, 2003.

[Wan2001] Wang, Rong, Feiyi Wang, and Gregory T. Byrd. "Design and Implementation of Acceptance Monitor for Building Scalable Intrusion Tolerant Systems",*Computer Communications and Networks Proceedings, Tenth International Conference*, 2001.

[Web2007] Web Services Policy 1.5 - Framework, W3C Recommendation 04 September 2007, http://www.w3.org/TR/ws-policy/. [Jan 31, 2014].

[Win2014] Windows Azure. http://www.windowsazure.com/. [Feb 01, 2014].

[Wol2014] "Wolfram Mathematica Online Integrator." *Wolfram Mathematica Online Integrator*. N.p., n.d. Web. 20 Jan. 2014. <http://integrals.wolfram.com>.

[Wyl2000] Wylie, J.j., M.w. Bigrigg, J.d. Strunk, G.r. Ganger, H. Kiliccote, and P.k. Khosla. "Survivable Information Storage Systems." *Computer* 33.8 (2000): 61-68.

[Zho2002] Zhou, L., F. Schneider, and R. Van Renesse. "Coca: A Secure Distributed On-line Certification Authority", *ACM Transactions on Computer Systems*, Nov. 2002.

BIOGRAPHY

Quyen L. Nguyen graduated as a valedictorian from the University of Delaware with a BS degree in Computer Information Sciences and Applied Mathematics. He obtained a MS degree in Computer Science from the University of California at Berkeley. He is currently a system architect at the US National Archives and Records Administration. Before joining the National Archives, he has worked for telecommunications software companies. His research interests include security architecture, system architecture, emerging technologies, Cloud Computing and Big Data. He has co-authored papers in Intrusion Tolerance and Service-Oriented Architecture published in magazines and workshop proceedings. In addition, he has presented papers related to large systems for digital archives and digital preservation at international conferences.