

Improving Resilience of SOA Services along Space-Time Dimensions

Quyen L. Nguyen and Arun Sood
International Cyber Center and Department of Computer Science
George Mason University
Fairfax, VA 22030, USA
{qnguyeng@gmu.edu, asood@gmu.edu}

Abstract—In Service-Oriented Architecture, a service contains a set of operations with openly defined input and output parameters. In addition to these operations and traditional QoS, offered services need to implement different levels of intrusion tolerance. Indeed, intrusion tolerance has been recently presented as part of the defense-in-depth solution in order to enhance security resilience for services, as a complement to the traditional intrusion prevention and detection. While satisfying functional requirements, a service also exposes its attack surface via published operations, protocols, and accessible data as an adverse side effect, which makes it susceptible to exploitation by malicious actors. The resulting question is - how can services fulfill and maintain their intrusion tolerance QoS (IT-QoS) for security resilience and rapid recovery in the face of hostile attacks. In this paper, we propose an approach to tune a service so that its attackability can be controlled and the IT-QoS guaranteed despite the exposed attack surface. Our approach relies on Self-Cleansing Intrusion Tolerance (SCIT), a recovery-based intrusion tolerance architecture combined with service-oriented programming constructs. A quantitative analysis using Semi-Markov Process modeling provides a mathematical foundation for compensating the expansion of a service’s attack surface by tuning SCIT system parameters.

Keywords – SCIT; Intrusion Tolerance; SOA; MTTR; MTTSF; Semi-Markov.

I. INTRODUCTION

Service-Oriented Architecture (SOA) is based on loosely coupled services. These services can be atomic, such as infrastructure and common services. They can also be composite services that are composed of atomic services or even other composite services. A service is designed with clearly defined functionalities that other services and applications can use. Along with the functional requirements, a service must satisfy non-functional requirements, among which security quality is a crucial one. Indeed, due to the inherent distributed nature of services communicating with each other via networks and open protocols, security quality is critical in SOA in order to ensure availability, integrity, and confidentiality of composed services and thus make them usable. On the other hand, since security attacks have become more and more sophisticated, a system cannot rely solely on intrusion prevention and detection for its security protection. Therefore, intrusion tolerance techniques and mechanisms should be part of the solution for securing computer information systems.

While fulfilling functional and business requirements, a service tends to expose its attack surface, i.e. its resources via open interfaces, as a side effect, which makes it susceptible to exploitation by malicious actors. The more operations and data a service provides, the more its attack surface increases. Thus, given this dichotomy between service enhancements and security guarantee, which is necessary for a dependable service operation, the problem is how to ensure that a service continues to fulfill its functionalities and maintain their published level of intrusion tolerance QoS (IT-QoS) at the same time.

The main contribution of this paper is two-fold: a) we introduce the time dimension to the concept of exposure by the service’s attack surface, which can be viewed as “space” dimensions to resolve the above dichotomy; b) we present a mathematical foundation for a scheme to improve service resilience with the capabilities of reducing its exposure to attacks along space-time dimensions, and leveraging service-oriented programming constructs of service composition. Our approach consists of compensating the undesired expansion of a service’s attack surface due to enhancements by utilizing Self-Cleansing Intrusion Tolerance (SCIT) and restricting the exposure time. In fact, it would be impractical to rely solely on mechanisms to minimize attack surface as proposed in [3][5]. For instance, a service realized by a COTS product has its attack surface pretty much established when the product was released. In some cases, other intrusion avoidance subsystems and/or COTS configuration can be used to reduce the effective attack surface at service deployment time. However, such approaches have their own limitations. An approach based on timed-recovery such as SCIT provides additional engineering tools to tune a service so that its attackability can be controlled, and the IT-QoS guaranteed despite the exposed attack surface. Moreover, service composition would allow protecting an important service by fronting it with another service controlled by SCIT to reinforce the resilience of the to-be protected service.

According to Strigini [14], resilience characterizes the extent of continuation of providing service in the face of “shocks”, which can be malicious attacks in the security domain. Strigini goes further to state that resilience also includes the capability to recover from adverse stimuli. In light of this notion of security with resilience, we propose to study here two well-known metrics of a service provided by the SCIT architecture: Mean Time To Security Failure (MTTSF) and Mean Time To Recover (MTTR) via a stochastic analysis based on Semi-Markov Process modeling. The former metric gives an assessment of the service’s resilience to respond to transactions and client requests; the latter fits into the notion of quick recovery in the case of failures.

The remainder of this paper is organized as follows. Section II provides the background of Attack Surface and Recovery-
based Intrusion Tolerance. In Section III, we show the stochastic analysis of an atomic service and a composite service protected by the SCIT architecture. Section IV discusses the space-time resilience concept, in light of the mathematical results obtained in Section III, and provides the calculations for reduced attack surface. The paper ends with a conclusion and avenue for future work in Section V.

II. BACKGROUND AND RELATED WORK

A. Attack Surface Concept

A service’s attack surface has been represented formally as an automaton [3][7]. Conceptually, an attack surface is composed of resource vectors, or attack surface dimensions. Classes of these resources vary in the literature, and generally consist of methods, processes, channels, protocols, data items, web pages, and access rights. Specific instances of these resources depend on the services or software products. The attack surface measurement is expressed as some form of aggregate of the measurements of all resource vectors. Each such measurement is either a simple count [2], or the ratio of damage cost over attack effort [5][7]. In all cases, functions have been devised to assign weight to each resource in order to account for the relative impact and criticality in terms of security. As an example, a service running with administrator privileges carries more weight than one requiring regular user authentication, since compromising the former can yield wider range of resources to the attacker.

Liu [5] views attack surface as an external system attribute, and has shown its correlation with internal software qualities such as complexity, and coupling between services. Consequently, Liu’s theory allows software designers to minimize a service’s attack surface by improving internal software attributes, such as reducing complexity and coupling.

Since our study deals with the Service level, and not with the resources used by a Service, a single aggregate value for attack surface metric suffices to be plugged into our analytical results. Options of an aggregate metric are the total sum of all measurements [3] or the Euclidean norm calculated from the resource vectors’ measurements [2].

Despite some variations in the computation of the attack surface metric, the general consensus is that intuitively the size of a service’s surface is adverse to its security. Hence, developers and administrators should strive to reduce the attack surfaces of system services and components during software construction or configuration.

Instead of the static conception as in most of the work so far, we propose a novel approach which views attack surface as dynamic and evolving over time, through software releases or various operational environments. With this dynamic concept, reducing a service’s attack surface to improve its resilience can be achieved by controlling its time dimension, i.e. its exposure window to the environment via the recovery-based intrusion tolerance SCIT architecture.

B. Recovery-based Intrusion Tolerance

Recently, there has been focus in using recovery-based intrusion tolerance to complement system defense-in-depth. The FOREVER service removes faults by proactive reconfiguration implemented by robust underlying components [13].

Paulo Sousa and his colleagues combined periodic system rejuvenation with reactive recovery [12]. Whereas rejuvenation is automatic, recovery occurs when the perceived threat exceeds a tolerable threshold that affects the system’s correct functioning.

Huang and Gosh [4] proposed MAS (Moving Attack Surface), which relies on building services with diversity at different layers of the architecture stack from Operating System, virtualization layer, application layer, etc. Their approach is also coupled with periodic cleansing like SCIT.

In our previous work, we have shown that using SCIT architecture, system designers and engineers can tune quantitatively the S-Reliability QoS of a service [9]. SCIT’s recovery-based mechanism consists of automatic and periodic cleansing of the servers running on a virtualization layer, which allows the instantiation of multiple servers with possible options of having different guest operating systems on a single host machine. In the SCIT architecture, the Central Controller has the principal role of managing and controlling all the nodes to be protected. Each of these nodes goes through a continuous cycle of a) Live Spare state, in which the node is pristine but offline; b) Active state during the exposure window, where the node is online to serve incoming requests; c) Grace Period state with pre-configured duration, where the node stops accepting new transaction requests, but completes processing of already queued requests; d) and Cleansing state, where the node is offline and undergoes full restoration to a known pristine state.

As described in [10] for the SOA adaptation, a “node” described above becomes a Service Container, which can be implemented by an application server where services are deployed and activated. A Service Container, running in a virtual machine, will host services requiring the same level L of IT-QoS, which is tied to a determined value for the exposure window. Thus, we have a conceptual hierarchy of entities in the system:

- Container Group associated to a level of IT-QoS; SCIT rotational cleansing is performed within one Container Group.
- Service Container belonging to a Container Group.
- Services hosted in a Service Container running on a VM. The Service Registry component added in [10] is required by SOA to publish services and their QoS parameters so that services can be discovered by consumers. Since the Service Registry is accessible by external clients over the network, it can present a vulnerability to the system. However, we can secure the Service Registry by hosting it on a separate Container to be managed through the same cleansing cycle as other service containers. Note that the Service Registry only contains public service’s metadata, e.g. the information contained in WSDL (Web Service Description Language) files. Internal control parameters that are critical to SCIT management such as service cleansing time, and exposure window values are persisted in a data store local to the Central Controller. For state management within SOA, SCIT’s implementation utilizes Network Attached Memory Terracotta product to provide sharing states between SCIT nodes [1].

This paper enhances the analytical framework in [8] further by introducing the attack surface factor. The combination of this spatial dimension and the temporal factor inherent to SCIT.
facilitates the analysis encompassing design patterns and mechanisms of service composition and service diversity to improve resilience. As a result, our study, based on a mathematical model, will enable an additional tuning tool for achieving representative QoS parameters of MTTSF and MTTR to solve the challenge posed by the attack surface of a service.

III. SERVICE INTRUSION TOLERANCE ANALYSIS

In this section, we will analyze the resilience of services protected with SCIT mechanism within the architecture described above. Our analysis is based on the methodology presented by Madan [6], and mainly uses Semi-Markov Chains whose states capture the behaviors of both the attacker and the service being studied. In each of the cases below, we will establish the Semi-Markov Process model that allows us to compute the two QoS parameters MTTSF and MTTR.

A. Atomic Service

First, we start with an atomic service, i.e. one that does not need to depend on other services in the SOA framework. The analysis of a SCITized atomic service is very similar to the one studied in [8] for a single system. We would like to use the simple case of an atomic service to set the baseline for subsequent discussion of composite services in SOA and their associated attack surfaces. The study of MTTR is also added, as we consider it as part of the parameter set to characterize a service’s resilience. A Service S protected by SCIT undergoes three states (Figure 1): Good (G), Attacked (A), and Failure (F). Service S enters state F when it is compromised due to a successful attack by some malicious actor. Note that we have collapsed states G and V identified in [8] to a single state G. Table 1 summarizes the transition probabilities between the states depicted in Figure 1. Service S can go from G to state A with probability $P_A$, which is the attack probability. From state A, service S can recover to good state G, due to the periodic cleansing and recovery scheme described in the architecture section above; $P_C$ is the probability to go to cleansing state from state A. Services under attack in state A can still operate; it might also be possible that an attack ceases without causing security failure to the Service. From the operational point of view, it is possible to merge states G and A. But, in our model, state A represents an intermediary step between state G and state F, and allows us to introduce the attack probability that is needed for subsequent reasoning in section IV.

![Figure 1. State Transition: Atomic Service.](image)

Table 1. Transition Probabilities: Atomic Service.

<table>
<thead>
<tr>
<th></th>
<th>G</th>
<th>A</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>$1 - P_A$</td>
<td>$P_A$</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>$P_C$</td>
<td>0</td>
<td>$1 - P_C$</td>
</tr>
<tr>
<td>F</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

MTTSF. To compute MTTSF, we consider state F as an absorbing state, according to Madan [6]. Thus, the sets of transient states $X_t$ and absorbing ones $X_s$ are respectively: $X_t = \{G, A\}$; $X_s = \{F\}$, and the transition matrix is:

$$Q = \begin{pmatrix} 1 - P_A & P_A \\ P_C & 0 \end{pmatrix}$$

Let $x = (x_G, x_A)$ be the row vector containing the number of visits and $h = (h_G, h_A)$ the row vector containing the mean sojourn times at the transient states in $X_t$. Then, MTTSF is given by the formula:

$$MTTSF = \sum_{i \in X_t} x_i h_i$$ (1).

With $q = (1, 0)$ being the starting vector, the number of visits can be found by solving the system of equations:

$$x = q + xQ$$ (2).

Plugging the solution for $x$ in Eq. (2) gives us the following expression for MTTSF:

$$MTTSF = \frac{h_G + P_A h_A}{P_A (1 - P_C)}$$ (3).

Differentiating MTTSF in Eq. (3) with respect to $P_A$ and $P_C$ yields that:

$$\frac{\partial (MTTSF)}{\partial P_A} < 0 \text{ and } \frac{\partial (MTTSF)}{\partial P_C} > 0.$$ (4)

Therefore, we can conclude that MTTSF increases if we can make $P_A$ smaller and $P_C$ larger.

MTTR. For computing MTTR, we do not consider state F as an absorbing state, so the transition matrix $P$ will encompass all states in $X = \{G, A, F\}$. Thus, $P$ will contain all entries listed in Table 1. Let $y = (y_G, y_A, y_F)$ be the row vector containing the steady-state probabilities of states G, A and F, and $h = (h_G, h_A, h_F)$ the row vector containing the mean sojourn times at the states in $X$. Then, MTTR is given by the formula:

$$MTTR = \frac{y h_F}{\sum_{i \in X} y_i h_i}$$ (4).

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

$$y = yP \text{ and } \sum_{i \in X} y_i = 1$$ (5).

Plugging the solution for $y$ in Eq. (4) gives us the following expression for MTTR:

$$MTTR = \frac{P_A (1 - P_C) h_F}{h_G + P_A h_A + P_A (1 - P_C) h_F}$$ (6).

Differentiating MTTR in Eq. (6) with respect to $P_A$ and $P_C$ yields that:

$$\frac{\partial (MTTR)}{\partial P_A} > 0 \text{ and } \frac{\partial (MTTR)}{\partial P_C} < 0.$$ (7)

Therefore, we can conclude that MTTR decreases if we can make $P_A$ smaller and $P_C$ larger.
B. Enabler Service

In his study of attack surface, Howard has the notion of attack target and enabler; the latter is exploited by malicious actor to get access to and compromise the aimed target [3]. With the SOA paradigm, we can have a service orchestration with enabler services and a target service. One example of such model is an application composed of a User Input Service, which validates all user input criteria in search requests, and the Search Service performing the actual search. In that case, every request transition to the Search Service must go through the User Input Service, which plays the role of the Enabler.

A Login Service preceding a Purchase Order Service is another example. If Single Sign On (SSO) is used, then one can get to all target services \( \{TS_1, TS_2, ..., TS_n\} \) under the SSO domain after passing the Login Service (LS) once. Then, the following model can be viewed as applicable to each pair (LS, TS\(_1\)), (LS, TS\(_2\)), ... , (LS, TS\(_n\)). Actually, a clearer example would be the Authorization Service, which is required to be executed each time any Target Service is requested.

The key characteristic of the composite service with Enabler as depicted by Figure 2 is that an attacker must compromise the enabler service S\(_1\) first before breaking into the target service S.

![Figure 2. State Transition: Target and Enabler Services.](image)

In the state transition diagram of Figure 2, the transition from state F to FA expresses the notion of enabler and target. Our discussion considers only one enabler service, but can be extended to the case where there are multiple enablers in the service orchestration.

Table 2 explains the states depicted in the diagram of Figure 2, and Table 3 contains the transition probabilities between states.

<table>
<thead>
<tr>
<th>State</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>Good state</td>
</tr>
<tr>
<td>A</td>
<td>S(_1) under attack</td>
</tr>
<tr>
<td>F</td>
<td>Security failure in S(_1)</td>
</tr>
<tr>
<td>FA</td>
<td>Security failure in S(_1) and S under attack</td>
</tr>
<tr>
<td>FF</td>
<td>Security failure in both S(_1) and S</td>
</tr>
</tbody>
</table>

Table 3. Transitions: Target and Enabler Services.

<table>
<thead>
<tr>
<th>State</th>
<th>G</th>
<th>A</th>
<th>F</th>
<th>FA</th>
<th>FF</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>1-(P_{A1})</td>
<td>(P_{A1})</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>(P_{C1})</td>
<td>0</td>
<td>1-(P_{C1})</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>F</td>
<td>1-(P_A)</td>
<td>0</td>
<td>0</td>
<td>(P_A)</td>
<td>0</td>
</tr>
<tr>
<td>FA</td>
<td>0</td>
<td>0</td>
<td>(P_C)</td>
<td>0</td>
<td>1-(P_C)</td>
</tr>
<tr>
<td>FF</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Our analysis focuses on the impact of the Enabler on the Target within this type of composite service. In other words, the viewpoint is on how the Enabler Service can provide security protection to the Target within the SCIT framework. Thus, states FA and FF signify more as the attack and failure states respectively of the Target Service.

In this section, we also focus on the resilience metrics of MTTSF and MTTR of service S within the composition. The analysis steps are very similar to the case of the atomic service, when only service S is being studied.

**MTTSF.** To compute MTTSF for service S, we consider state FF as the absorbing state, so the sets of transient states \(X_t\) and absorbing states \(X_a\) are respectively: \(X_t = \{G, A, F, FA\}\); \(X_a = \{FF\}\). Thus, the transition matrix \(P\) will take entries from Table 3 above, excluding the last row and column pertaining to state FF.

The row vectors \(\mathbf{x}\), \(\mathbf{h}\) and \(\mathbf{q}\) will be extended to \(\mathbf{x} = (x_G, x_A, x_F, x_{FA})\), \(\mathbf{h} = (h_G, h_A, h_F, h_{FA})\) and \(\mathbf{q} = (1, 0, 0, 0)\) respectively, since \(X_t\) contains more states. Eq. (1) can still be applied to MTTSF with new values of \(\mathbf{x}\), \(\mathbf{h}\), and \(\mathbf{q}\).

Furthermore, plugging the solution for \(x\) in Eq. (2) into Eq. (1) yields the expression for the mean time to security failure of service S in the composite case:

\[
MTTSF = h_G + h_A P_{A1} + P_{A1} (1 - P_{C1}) h_F + h_{FA} P_A (1 - P_{A} P_{C}) \tag{7}
\]

Differentiating MTTSF in Eq. (10) with respect to \(P_{A1}\), \(P_{C1}\), \(P_A\) and \(P_C\) yields the following:

\[
\frac{\partial(MTTSF)}{\partial P_{A1}} < 0; \quad \frac{\partial(MTTSF)}{\partial P_{C1}} > 0; \quad \frac{\partial(MTTSF)}{\partial P_A} < 0; \quad \text{and} \quad \frac{\partial(MTTSF)}{\partial P_C} > 0;
\]

This leads us to the same conclusion as for atomic service that MTTSF increases if we can make \(P_A\) and \(P_{A1}\) smaller, \(P_C\) and \(P_{C1}\) larger.

**MTTR.** Solving the adapted Eq. (5) and plugging the solution for \(x\) in Eq. (4) gives us the expression for MTTR, with the conditions that \(\mathbf{y} = (y_G, y_A, y_F, y_{FA})\), \(\mathbf{h} = (h_G, h_A, h_F, h_{FA})\) and the transition matrix \(P\) containing all entries listed in Table 3 above:

\[
MTTR = \frac{P_{A1} P_{C1} P_A P_C h_{FF}}{(h_G + h_A P_{A1}) (1 - P_{C1}) + P_{A1} P_{C1} [h_F + P_A (h_{FA} + h_{FF} P_C)]}\tag{8},
\]

where \(P_{C1} = (1 - P_C)\) and \(P_{C1} = (1 - P_{C1})\).

Differentiating MTTR in Eq. (8) with respect to \(P_A\) and \(P_C\) yields that:

\[
\frac{\partial(MTTR)}{\partial P_{A1}} > 0; \quad \frac{\partial(MTTR)}{\partial P_{C1}} < 0; \quad \frac{\partial(MTTR)}{\partial P_A} > 0 \quad \text{and} \quad \frac{\partial(MTTR)}{\partial P_C} < 0.
\]
Therefore, MTTR will decrease if we can make $P_A$ and $P_{A_1}$ smaller, $P_C$ and $P_{C_1}$ larger.

C. Pass-through Service

The paths formed by the transitions $G_1 \rightarrow G_2 \rightarrow A_2$ and $F_1 \rightarrow G_2$ in Figure 3 mean that services $S_1$ and $S_2$ are independent when it comes to being attacked. In fact, the utilization of the methods in previous sections gives inconclusive results as to the impact of $P_{A_1}$ and $P_{C_1}$ of Service $S_1$ onto the MTTSF and MTTR 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 ($P_{A_1}$, $P_{C_1}$) and ($P_{A_2}$, $P_{C_2}$) 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.

![Figure 3. State Transition: Pass-through Service.](image)

IV. SPACE-TIME RESILIENCE SCHEME

A. Service Space-Time Dimensions

A service $s$ in operation in the SCIT environment has two attributes, namely its attack surface metric $\sigma$ and its exposure window $\mu$. From another perspective, we want to introduce the notion of effective attack surface metric $\sigma_E$, which represents the actual usable value of the attack surface present in our analysis. $\sigma_E$ may very well be the same as $\sigma$ as in the case of an atomic service; but it is not necessarily true for other cases. Since $\sigma_E$ can change over time due to service upgrades, patches and configuration changes, and $\mu$ may have to be adjusted accordingly, both attributes can be expressed as step functions of time $t$:

$$s = (\sigma_E(t), \mu(t)).$$

B. Atomic Service

The attack surface metric $\sigma$ of service $S$ can be evaluated using the ratio of potential damage over required effort [7]. Taking into account the attack surface metric and the arrival probability of attacks, we can evaluate the attack probability $P_A$ as:

$$P_A = P_R P_S$$

In Eq. (9), $P_A$ is the probability that an arrived attack targets the attack surface of service $S$. Furthermore, we can see from Eq. (9) that one can lessen the impact of $P_S$ by reducing arrival of attack $P_R$. Practically, the software implementation has the trend to add more and more functionalities to a given service over time, in order to adapt to new environment and increasing demand of the users. A side effect of these enhancements could be that the attack surface of a new release of that service is larger than that of the old release. So, one application suggested by Eq. (9) is to contract $P_S$ in order to compensate for the expansion of a service’s attack surface. In our earlier work [8], we have established the result that $P_R$ is controlled by the SCIT exposure window $\mu$. Assuming that attack arrival is a Poisson process with rate $\lambda_1$ as suggested in [6], we can write:

$$P_R = 1 - e^{-\lambda_1 \mu}$$

With care taken to compute $\sigma$ such a way that the attackability of any surface unit is the same (i.e., the expected rate of attack occurrences per surface unit is fixed to $\lambda_2$), we can model $P_S$ by an exponential distribution:

$$P_S = 1 - e^{-\lambda_2 \sigma}$$

Using (9), (10) and (11), we obtain the effective attack surface of an atomic service protected by SCIT as:

$$\sigma_E = \frac{1}{\lambda_2} \ln \left( \frac{1 - e^{-\lambda_1 \mu}}{e^{-\lambda_2 \sigma}} \right)$$

The effective attack surface $\sigma_E$ expressed in (12) can be easily proved to decrease as $\mu$ decreases when SCIT is used. Without SCIT, i.e., $\mu = \infty$, $\sigma_E$ is simply equal to the original $\sigma$.

C. Enabler Service

In the case of a composite with enabler and target services, the state transition in Figure 4 suggests that we can make:

$$P_A = 1 - P_{C_1}$$

Eq. (13) is quite interesting, as it shows that we can control the attack probability to service $S$ via the cleansing probability $P_{C_1}$ of the enabler service $S_1$. Indeed, $P_A$ can be made smaller – hence improving $S$’s resilience -- by increasing $P_{C_1}$. Since we have $P_{C_1} \geq e^{-\lambda_1 \mu_1}$ as argued in [8], the probability that the composite service in state $F$ goes to cleansing mode can be made larger by decreasing the exposure window $\mu_1$ of the enabler service $S_1$. Then, $S$’s resilience is protected by $S_1$, and the impact of $S$’s attack surface is greatly diminished. From (10), (11) and (13), we can derive the upper bound for the effective attack surface of the target service $S$ as:

$$\sigma_E \leq \frac{1}{\lambda_2} \ln \left( \frac{1 - e^{-\lambda_1 \mu}}{e^{-\lambda_2 \sigma}} \right)$$

In the operational environment, since $\frac{d \sigma_E}{d \mu_1} > 0$ iff $\mu_1 > \mu$, the enabler service must be configured to have exposure window smaller than that of the target in order to exploit the protection by the enabler service.

D. Diversity

In the literature of Intrusion Tolerance, diversity has been proposed as one of the mechanisms to increase the resilience of services. In this section, we will establish probabilistic relations between the diversity of a service and its resilience. Let $\{S_i\}_{i=1,..,N}$ be the set of $N$ different versions of the same service $S$, which can be obtained by using different operating systems, middleware packages, or implementations as proposed by Huang [4]. Assuming that a line of attack only works for a specific version of the service $S$, the attack probability of the surface of $S$ is bounded by:

$$P_S \leq \frac{1}{N} \left\{ \max_{i=1,..,N} P_{S_i} \right\}$$

According to (15), the probability of attack on the service’s surface will decrease if the system uses more diverse versions of the same service; the factor $(1/N)$ expresses the probability...
of a match between a line of attack against a specific version of the service. Using Eq. (11), we can express the upper bound for $P_S$ in terms of attack surface metrics:

$$P_S \leq \frac{1}{N} \left(1 - e^{-\lambda_2 \sigma_{\text{max}}} \right)$$  \hspace{1cm} (16),

where $\sigma_{\text{max}}$ is the maximum attack surface metric of all versions $s_i$. Next, we undertake the expected value analysis. Let us consider $\sigma(t)$ as a random variable whose state space is the set $\{s_i\mid i=1,...,N\}$ where $\sigma_i$ is the attack surface metric of version $s_i$. If each version of the service $S$ is chosen with equal probability by SCIT Controller, then, the expected value of $\sigma$ is:

$$E[\sigma] = \sum_{i=1}^{N} \frac{1}{N} \sigma_i = \bar{\sigma}$$  \hspace{1cm} (17),

Thus, using the reasoning underlying Eq. (16), the average value for $\sigma$, and combining with Eq. (11), $P_S$ becomes:

$$P_S = \frac{1}{N} \left(1 - e^{-\lambda_2 \bar{\sigma}} \right)$$  \hspace{1cm} (18).

Eq. (18) has great significance since it shows that service diversification decreases the probability of attack on the service's surface by a factor of $(1/N)$. The effective attack surface becomes then:

$$\sigma_{\text{e}} = \frac{1}{\lambda_2} \ln \left(\frac{N}{N + e^{-\lambda_2 \bar{\sigma}} - 1}\right)$$  \hspace{1cm} (19),

which approaches to 0 as $N \to \infty$. The interpretation of this result is that if diversity were used in SCIT, then the service’s effective attack surface would be drastically reduced. It was pointed out in [4] that there is quite a cost and complexity for managing diverse versions of a service, especially when diversity is implemented at all layers of the service stack. However, we have shown in [11] that diversity can be easily realized in the case of C-SCIT, a cloud-based version of SCIT.

E. Summary

In (3), (6), (7), and (8), we have obtained the expressions of MTTSF and MTTR in terms of $P_C$, $P_a$, and $P_S$, which represent the probabilities of cleansing, attack arrival, and successful attack to a surface respectively in the case of atomic and composite services with the target-enabler model. With appropriate assumptions about attack arrival and attack success on a service's surface, we can show that MTTSF and MTTR are functions of the attack surface $\sigma$ and exposure window $\mu$, so that we can write:

$$\text{MTTSF} = F(\sigma, \mu)$$  \hspace{1cm} (20a) \hspace{1cm} \text{and} \hspace{1cm} $$\text{MTTR} = G(\sigma, \mu)$$  \hspace{1cm} (20b).

With either Eq. (20a) or (20b), one can find $\sigma$ and $\mu$ to satisfy a desired value for MTTSF or MTTR. Normally, $\sigma$ is known at service deployment time, so $\mu$ is solvable.

V. CONCLUSION AND FUTURE WORK

With the SCIT architecture adapted to SOA environment, our study based on stochastic process analysis reveals three approaches to improve service’s resilience:

1. Compensating the attack surface by smaller exposure window;
2. Reducing the exposure window of Enabler Services in a service orchestration;
3. Reducing the effective attack surface of a service by having multiple diverse configurations of that service.

In this paper, we have not discussed about the cost of securing services via SCIT. The number of replicas needed to realize a required value of exposure window is calculated in [9]; this can be combined with a service’s system requirements to derive computing resources cost. According to the testing reported in [1], the overhead of SCIT mechanism is minimal as reflected by the response time measurements. One future work could be to establish an analytical model for SCIT performance impact. The concept of tuning along space (attack surface) and time (exposure window) presented in this paper centers around the Service entity. An attack surface is represented by vectors capturing elements below the service level, so applying concepts of space and time developed in this paper to lower level entities deserves further consideration.

ACKNOWLEDGEMENT

We thank the reviewers for their comments which helped us improve the quality of this paper.

REFERENCES


