Building a Resilient Service-Oriented Architecture Environment

Quyen L. Nguyen, International Cyber Center and Department of Computer Science
Arun Sood, George Mason University and SCIT Labs

Abstract. In a Service-Oriented Architecture (SOA) system, services contain operations with openly defined input and output parameters. While satisfying functional requirements, a service also exposes its attack surface via published operations, open protocols, and accessible data as an adverse side effect, which makes it susceptible to exploitation by malicious actors. With this context, it is a challenge to build an SOA environment such that it is resilient in the face of hostile attacks. In this paper, we propose an approach to design services such that their attackability can be controlled and intrusion tolerance 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.

1. Introduction

SOA is based on loosely coupled services. Services such as infrastructure and common services are atomic. Other services are called composite services because they 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 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 systems should be part of the solution for securing computer information systems.

The challenge of making a service resilient is that it is bound to expose resources via open interfaces as a side effect while fulfilling functional and business requirements. Each resource is likely to have vulnerabilities, and these together constitute the service's attack surface, which includes operations, data items accessible to read and/or write, and communication channels over which services operate [1]. The more operations and data a service provides, the more its attack surface increases. Thus, limiting the attack surface does not always work. Moreover, if COTS is used to realize a service, then the service's attack surface is pretty much predefined, leaving little room for reducing its exposure via configuration.

In this paper, we will present an approach to build a resilient SOA environment, which has four characteristics: a) to withstand malicious attacks; b) to rapidly recover from compromises; c) to provide service continuity even in the case of attacks; and d) to adapt to changing operational environment. The characteristics a) and b) can be quantitatively represented by the QoS parameters of Mean Time to Security Failure (MTTSF), and Mean Time to Repair (MTTR) respectively. 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 [2]. Characteristic c) of service continuity can be mitigated by having redundant live nodes, with the pair of primary/backup and diversity in the SCIT environment. We will also show how SCIT allows services to adapt to changing environment while maintaining their promised resilience. Based on timed-recovery mechanisms provided by SCIT, software architects are offered additional tools to design a service so that its attackability can be controlled, and intrusion tolerance QoS guaranteed despite the exposed attack surface. Moreover, our approach leverages service composition constructs that allow fronting an important service with another service well controlled by SCIT to reinforce the resilience of the service to be protected.

2. System Architecture

SCIT Mechanism

The goal of SCIT is to make applications and services resilient in the face of malicious attacks. 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. Moreover, the utilization of virtual machines enables rapid reloading and reactivation of the servers. SCIT's pattern operates on two major components as depicted in Figure 1:

a. Central Controller managing and controlling all the nodes to be protected. Diversity of these nodes can be employed to further reduce the likelihood of malicious exploitations. For example, we can have a group of web servers, one running on Linux, another on Windows, and a third one on Mac OS.

b. Cluster of nodes providing the same applications and services. Managed by the Central Controller, each node continuously goes through the following state sequence:

- Live Spare state, in which the node is pristine but offline;
- Active state for the duration W_o, (called exposure window), where the node is online to serve incoming requests;
- Grace Period state with pre-configured duration, where the node stops accepting new transaction requests, but completes processing of already queued requests;
- Cleansing state, where the node is offline and undergoes the full restoration to a known pristine state.

Given that the cleansing time depends on the specific service, we have obtained the number of redundant nodes in a cluster to perform the rotation cycle of duration W_o [2]. Figure 1 shows that Node 1 is in Active state, Node 2 in Live Spare state, and Node n in Cleansing state.

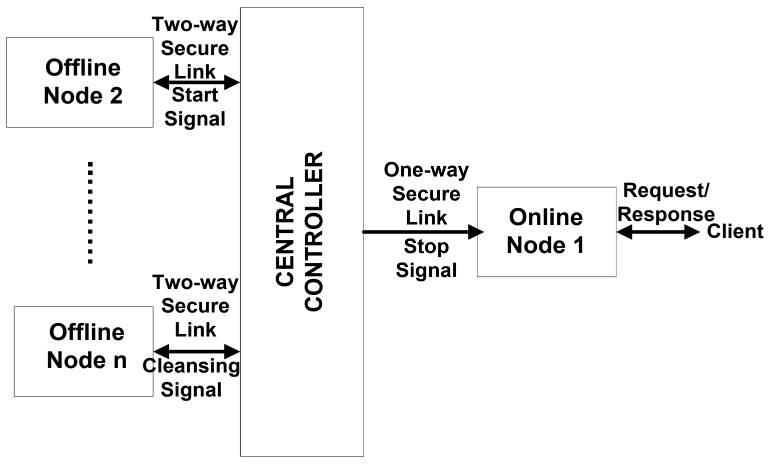

Figure 1. SCIT Architecture Components.

In terms of deployment, it is advisable to host the Controller physically separated from the nodes in order to avoid a data path between the nodes and the Controller, thus eliminating potential malware propagation to the Controller. Furthermore, in the current SCIT pattern, the link from the Controller to the online node is unidirectional.

The SCIT product has been developed by SCIT Labs under the direction of Dr. Sood. Experimentation was performed on the implementation of SCIT and reported in [3], where results demonstrated that the performance and computing resource usage impacted by SCIT is minimal. Recently, SCIT Labs was selected for a Small Business Innovation Research award that focuses on Scalable Moving Target Defense based on SCIT technology. Moreover, they plan to perform more tests as to the application of SCIT in various domains—defense, civil, and commercial. Such tests would assess the performance and the security of the implementation.

Cleansing Mode

The main functionality of SCIT is node cleansing, which consists of bringing a node back to a pristine state based on a virtual image stored and maintained in a safe place. The virtual image of a service can be created on an offline machine to ensure its unreachability by potential hackers. There are two cleansing modes to accommodate two types of services:

Full cleansing mode is used for services with transactional operations. Usually, those transactions are short requests so that they can be implemented as stateless RESTFul web services. In this basic cleansing mode, the nodes will undergo full erasure to make room for the clean image. This image is static because it does not contain any in-flight data. In fact, since the service operations are stateless, there is no need to capture state information of the node before performing the cleansing.

Partial cleansing mode is used for services with long running operations. Scientific computation falls into this category. Since state information may exist when a node enters the cleansing phase, using full cleansing mode may unintentionally cause loss of process state information, hence corrupting the on-going computation. This partial cleansing mode includes the following steps:

- Step 1. Capture the state of the service running in the node running the virtual machine, and create a snapshot dynamic image. The running state of the service is comprised of resources such as memory, file descriptors, buffer content, program counter, stack pointers, registers, etc. used by the service in the virtual machine. There has been research done in this area. Aroma [4] is a Java compatible VM allowing the capture of state and threads. "Process Introspection" was proposed by Ferrari to capture process state and perform its recovery [5].
- Step 2. Perform cleansing.
- Step 3. Migrate the snapshot containing the service's state to the node scheduled to be turned online.

3. Resilient Service Design

In this section, we will analyze the relationship between SCIT exposure window and the resilience of services expressed by the QoS parameters MTTSF (Mean Time To Security Failure) and MTTR (Mean Time To Repair), via the methodology presented in [6]. The analysis utilizes Semi-Markov Chain whose states capture the behaviors of both the attacker and the service being studied. As a result of the established correlation, designing a service's resilience is tantamount to computing the exposure window and the number of diverse and redundant services. To confirm this theoretically proven mechanism of improving resilience QoS parameters of MTTSF and MTTR, testing will be performed in addition to the experiments reported in [3].

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. Figure 2 shows that service S protected by SCIT undergoes three states: Good (G), Attacked (A), and Failure (F). An atomic service starts with state G. In the case where the service is attacked by some malicious actor, it transitions to state A with attack probability P_A. Service S enters state F when it is compromised. Thanks to the SCIT periodic cleansing and recovery scheme, service S can recover to good state G from state A, with probability P_C. Similarly, service S can get out of state F to go to state G because of SCIT forced cleansing.

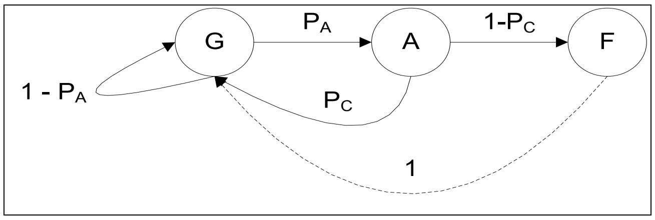

Figure 2. SCIT Atomic Service Transition Diagram.

P_A is factored by the service's attack surface and attack arrival. In [2], we have proved that the attack arrival and P_C can be controlled by exposure window W. Since MTTSF and MTTR are conditioned by P_A and P_C, it can be shown that MTTSF increases and MTTR decreases when W decreases.

Orchestration with Dependent Services

The notion of attack target and enabler is described in [7]; malicious intruders can exploit the enabler's attack surface to get access to and compromise the aimed target. With the SOA paradigm, we can have a service orchestration with enabler services and a target service. For example, let us consider the case of an archive application that allows a public user to search for digital records preserved in the archive. The application is implemented as a composition of User Input Service, which validates all user input criteria in the search requests, and the Search Service performing the actual search. The Search Service can be based on a Free Open Source Software or COTS, such as Apache Solr, SeekQuarry, Autonomy, Vivisimo, Google Search Appliance, FAST, etc., whose attack surfaces have been pre-determined at release time. The key characteristic of the composite service with enabler as depicted by Figure 3 is that an attacker must compromise the enabler service S1 first before she can break into the target service S.

States G, A, and F of enabler service S_1 are the same as for an atomic service. The transition from state F to FA expresses the notion of enabler and target mentioned above. State FF happens when both services are compromised. Our discussion considers only one enabler service, but can be extended to the case where there are multiple enablers in the service orchestration.

Since the composite service can transition from F to G thanks to cleansing, we can make the following (see Equation 1):

$$P_A = 1 - P_{C1}.$$

Equation 1:

The significance of this equation is quite interesting. Indeed, it reveals that probability of attack to the target can be controlled by means of the cleansing probability P_{C1} of the enabler service S_1. Overcoming S's attack surface, hence improving its resilience can be realized by increasing P_{C1}.

Orchestration with Independent Services

Services S_1 and S_2 are independent in terms of attackability, as shown by the paths $G_1 \rightarrow G_2 \rightarrow A_2$ and $F_1 \rightarrow G_2$ (Figure 4). The same would apply for parallel or conditional branches in a service orchestration. Consequently, in order to analyze these cases, we can apply the results for an atomic service to the services S_1 and S_2 separately.

Diversity

In the literature, diversity has been proposed as one of the mechanisms to increase the resilience of services. Let $\{s_i \mid i=1,..,N\}$ 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 [8]. Assuming that a line of attack only works for a specific version of a service S, the attack probability P_S of the surface of S is bounded by the following (see Equation 2):

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

Equation 2:

In Equation 2, the values P_{Si} are the attack probabilities of the different version of the same service S. This expression shows that the probability of attack on the service's surface will decrease if more versions are used.

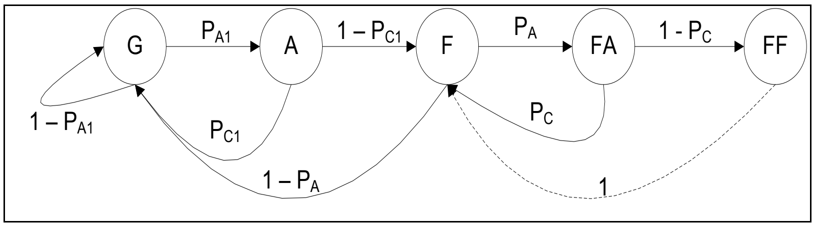

Figure 3. State Transition of Dependent Services.

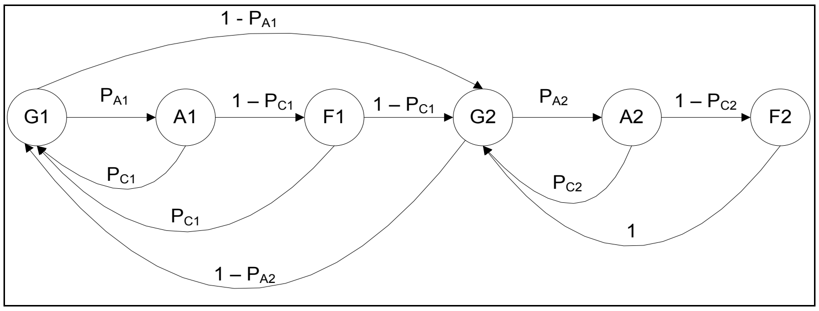

Figure 4. State Transition of Independent Services.

4. SCIT and SOA

Service Provider Architecture

For a single service provider, the SCIT architecture depicted in Figure 1 is enhanced with two additional components (Figure 5) as described in [9]:

- The Service Registry contains metadata about the services in the system including service operations, access points, and Intrusion Tolerance QoS (IT-QoS) characteristics. The notion of IT-QoS was introduced in [9] to capture QoS attributes that are related to an intrusion tolerant system, such as MTTSF, S-Reliability, MTTR, maximum intrusion number, etc. For scientific computation, Computational QoS needs to be considered [10]. The Central Controller queries the Service Registry about desired IT-QoS values for a service, and also updates the current operational values.
- The Monitor provides the Central Controller with information about the current operating environment based on analytics of logs, or reports provided by sensors such as IDS sensors.

In the SOA adaptation, a "node" in the SCIT architecture pattern described above becomes a Service Container, which can be implemented by an application server where services are deployed and activated. A Service Container will host services requiring the same level L of IT-QoS, which is tied to a value for the exposure window. In order to apply SCIT periodic cleansing, the system needs N replicas of similar service containers, i.e. containers with services of same IT-QoS level L. These N replicas form a Container Group associated with a specific level L.

A system providing differentiated IT-QoS levels will have multiple Container Groups. Figure 2 exhibits an example of a system with 4 services S1, S2, S3 and S4. There are only two levels of IT-QoS denoted by level 1 and level K. Due to the low level of level 1, Group 1 only needs 2 replicas of Service Containers, namely Containers 11 and 12. For the higher level K, 3 Service Containers are configured: Containers K1, K2 and K3. Note that services S1 and S3 are contained in all containers, since they operate at both levels 1 and K. But, S2 instances are deployed only in containers of level 1, while S4 instances only in containers of level K.

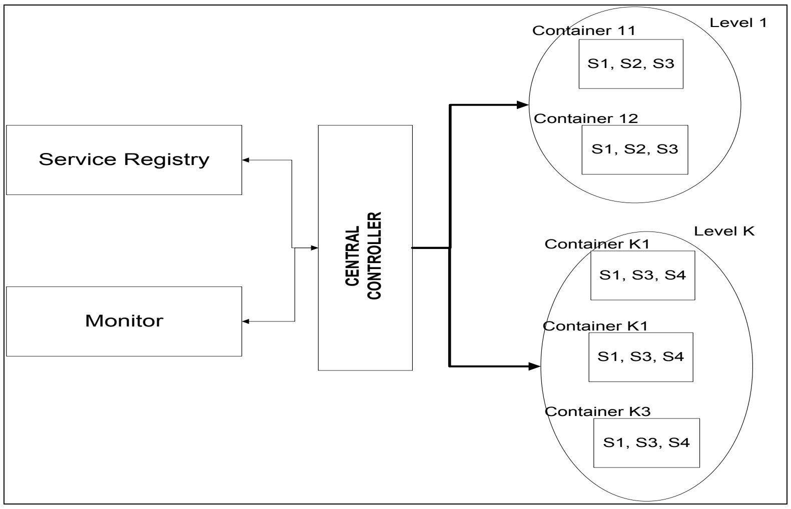

Figure 5. SCIT for a Single Service Provider's Services.

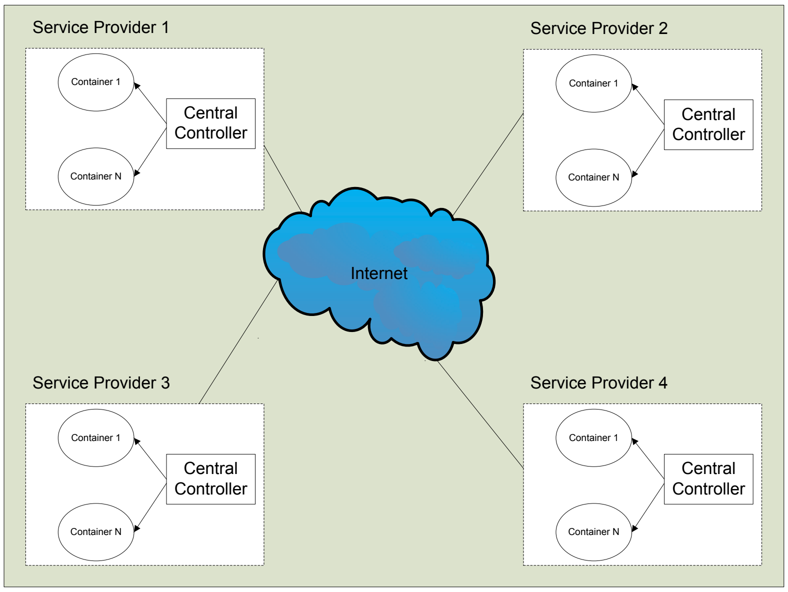

Figure 6. Resilient SOA Ecosystem.

Resilient SOA Ecosystem

In SOA, services can be provided by various providers. For instance, an application can utilize storage service from Amazon S3, and incorporate Google map service. If we can apply the above SCIT resilient architecture to the service providers involved in building an SOA application, then we can create a resilient SOA ecosystem. Figure 6 shows four service providers whose environments have implemented SCIT. Each service provider will have its own Central Controller to manage the cleansing of its service containers.

5. Conclusion

In this paper, we have shown that the SCIT architecture of Controller, Service Container, and cleansing algorithm based on exposure window contributes to build a resilient SOA environment. This is achieved by increasing each service's resilience in the SOA ecosystem by compensating the attack surface with a smaller exposure window, reducing the exposure window of Enabler Services in a service orchestration, or reducing the effective attack surface of a service with multiple diverse configurations of that service. Thanks to the simple parameters of exposure window and diversity number, software architects can specify resilience quality quantitatively during the service design.

REFERENCES

1. Pratyusa K. Manadhata and Jeannette M. Wing. "An Attack Surface Metric". IEEE Transactions on Software Engineering, May-June .2011.<http://testlab.sit.fraunhofer.de/downloads/Publications/heumann-quantifying_the_attack_surface_of_a_web_application-GI_Sicherheit_2010.pdf>
2. Quyen Nguyen and Arun Sood. "Quantitative Approach to Tuning of a Time-Based Intrusion-Tolerant System Architecture". WRAITS 2009, Lisbon, Portugal. <http://wraits09.di.fc.ul.pt/wraits09paper2.pdf>.
3. Anantha Bangalore and Arun Sood. "Securing Web Servers using Self Cleansing Intrusion Tolerance (SCIT)". Proceedings of Second International Conference on Dependability (DEPEND 2009), Athens, Greece. June 18-23, 2009.
4. Niranjan Suri , Jeffrey M. Bradshaw , Maggie R. Breedy , Kenneth M. Ford , Paul T. Groth , Gregory A. Hill , Raul Saavedra. "State capture and resource control for java: The design and implementation of the aroma virtual machine". In Proceedings of the Java Virtual Machine Research and Technology Symposium, 2001.
5. Adam Ferrari, Steve J. Chapin and Andrew Grimshaw. "Heterogeneous process state capture and recovery through Process Introspection". Journal of Cluster Computing, Volume 3, Issue 2, 2000.
6. Bharat B. Madan, Katerina Goseva-Popstojanova, Kalyanaraman Vaidyanathan, and Kishor S. Trivedi. "A Method for Modeling and Quantifying the Security Attributes of Intrusion Tolerant Systems". Dependable systems and networks-performance and dependability symposium (DSN-PDS) 2002.
7. Michael Howard, Jon Pincus, and Jeannette M. Wing. "Measuring Relative Attack Surfaces". <http://www.cs.cmu.edu/~wing/publications/Howard-Wing03.pdf>.
8. Yih Huang and Anup K. Ghosh. "Introducing Diversity and Uncertainty to Create Moving Attack Surfaces for Web Services". Advances in Information Security, 1, Volume 54, Moving Target Defense, Pages 131-151.
9. Quyen Nguyen and Arun Sood. "Realizing S-Reliability for Services via Recovery-driven Intrusion Tolerance Mechanism". 2010 International Conference on Dependable Systems and Networks Workshops (DSN-W), Chicago, Illinois. Jun 28-Jul 1, 2010.
10. Boyana Norris , Jaideep Ray , Rob Armstrong , Lois C. Mcinnes , Sameer Shende. "Computational Quality of Service for Scientific Components". Proceedings of the International Symposium on Component-based Software Engineering (CBSE7).

ABOUT THE AUTHORS

Quyen L. Nguyen is a system architect in the Systems Engineering Division of the Electronic Records Archives Program Management Office at the US National Archives and Records Administration. His research interests include system architecture, software modeling, and security architecture. Nguyen has a BS in computer and information sciences from the University of Delaware, an MS in computer science from the University of California at Berkeley, and is a computer science PhD candidate at George Mason University.

E-mail: qlnguyen@yahoo.com

Dr. Arun K. Sood is Professor of Computer Science in the Department of Computer Science, and Director of the International Cyber Center (ICC) at George Mason University, Fairfax, VA. His research interests are in security architectures; image and multimedia computing; performance modeling and evaluation; simulation, modeling, and optimization.

He and his team of faculty and students have developed a new approach to server security, called Self Cleansing Intrusion Tolerance (SCIT). We convert static servers into dynamic servers and reduce the exposure of the servers, while maintaining uninterrupted service. This research has been supported by US Army, NIST through the Critical Infrastructure Program, SUN, Lockheed Martin, Commonwealth of Virginia CTRF (in partnership with Northrop Grumman).

Recently SCIT technology was winner of the Global Security Challenge (GSC) sponsored Securities Technologies for Tomorrow Challenge. This technology has been awarded 3 patents and 3 additional patents are pending. SCIT Labs, a university start up, has been formed to commercialize SCIT technology – Dr. Sood is the founder and CEO of SCIT Labs.

Since 2009 Dr. Sood has directed an annual workshop on Cyber Security and Global Affairs with Office of Naval Research support – Oxford 2009, Zurich 2010, Budapest 2011 and Barcelona 2012.

Dr. Sood has held academic positions at Wayne State University, Detroit, MI, Louisiana State University, Baton Rouge, and IIT, Delhi. His has been supported by the Office of Naval Research, NIMA (now NGA), National Science Foundation, U.S. Army Belvoir RD&E Center, U. S. Army TACOM, U.S. Department of Transportation, and private industry.

He was awarded grants by NATO to organize and direct advance study institutes in relational database machine architecture and active perception and robot vision.

Dr. Sood received the B.Tech degree from the Indian Institute of Technology (IIT), Delhi, in 1966, and the M.S. and Ph.D. degrees in Electrical Engineering from Carnegie Mellon University, Pittsburgh, PA, in 1967 and 1971, respectively.

His research has resulted in more than 170 publications, two edited books, 8 patents, and his resume including publications list is available at http://cs.gmu.edu/~asood.

Professor, Computer Science & Director, International Cyber Center
George Mason University
MS 4A5, 4400 University Drive, Fairfax, VA 22030
E-mail: asood@gmu.edu

Founder and CEO SCIT Labs Inc
13834 Springstone Dr., Clifton, VA 20124
E-mail: asood@scitlabs.com