Biologically Inspired Security as a Service for Service-Oriented Middleware

Tashreen Shaikh Jamaluddin[1], Hoda Hassan[2] and Haitham S. Hamza[3]
[1]*Computer Science Department, AASTMT Academy, Qism El-Nozha, Cairo, Egypt*
[2]*Electrical Engineering Department, British University in Egypt, ElShrouk, Cairo, Egypt*
[3]*Information Technology Department, Cairo University, Ahmed Zewail st., Cairo, Egypt*

Keywords: Security as a Service (SECaaS), Denial-of-Service (DoS), Service-Oriented Architecture (SOA), Service-Oriented Middleware (SOM).

Abstract: Service-Oriented computing is a new programming paradigm based on service-oriented architecture that uses web services as its basic building block. Service-Oriented Middleware is a middleware layer that was developed to support service-oriented computing by allowing the flexible integration and operation of web services within the service-oriented computing environment. With the wide adoption of service-oriented computing, web service applications are no longer contained within tightly controlled environments, and thus could be subjected to malicious attacks, such as Denial of Service attacks. In this paper, we propose a generic security service that protects web services against denial of service attacks at the service-oriented middleware layer. Our security service draws on a bio-inspired framework that was developed to counteract denial of service at the network layer. To evaluate our work we have developed a prototype that showed that our proposed security service was able to detect denial of service attacks targeting a web service.

1 INTRODUCTION

Service Oriented Architecture (SOA) is a form of distributed system architecture for developing and integrating enterprise applications (Jensen et al. 2007; Jensen et al. 2009). It enables an enterprise to expose software components as self-describing, loosely coupled, coarse-grained, and re-usable business functions. SOA, with its loosely coupled nature, is widely used to provide interoperable services and to reuse existing services. Within SOA, application functionalities can be integrated and invoked with a variety of platform-independent service interfaces available through standard network protocols.

A Web Service is an application that is described, published, and invoked over the Web through an identifying URI (Bichler and Lin. 2006). Web service protocols and technologies include: XML (W3C 2008), XML Schema (W3C 2001), Web Services Description Language (WSDL) (W3C 2001), Universal Discovery Description and Integration (UDDI) (OASIS 2015) and Simple Object Access Protocol (SOAP) (W3C 2007). The web service public interfaces and bindings are defined in a WSDL document using XML. SOAP is a communication protocol that runs between the Service Requestor and the Service Provider. In most SOA applications, SOAP is adopted to develop Web services as SOAP is highly extensible and ensures confidentiality and integrity as specified within the WS-Security standards (Gudgin et al. 2007). The WS-Security protocol is the standard that is most widely used to implement an end-to-end security solution. Nevertheless, as noted in (WS-Security 2015), intrusion vulnerabilities can extend to XML-related processing, as well as to WS-Security web services standards.

With Web services, software applications began to be constructed based on independent services with standard interfaces. This led to huge distribution and heterogeneity in the software produced and in the communication technologies used. Thus interoperability was introduced as a new challenge in SOA based applications (Al-Jaroodi and Mohamed 2012). This situation motivated the development of the Middleware layer to handle the flexible integration and scalable interoperation between different heterogeneous platforms. Similar to traditional distributed middleware, Service

Jamaluddin, T., Hassan, H. and Hamza, H.
Biologically Inspired Security as a Service for Service-Oriented Middleware.
DOI: 10.5220/0006337801210132
In *Proceedings of the 12th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2017)*, pages 121-132
ISBN: 978-989-758-250-9

Oriented Middleware (SOM) acts as a software layer that abstracts the distribution of the underlying entities, integrates components and services, and provides common non-functional values to SOA based applications (Al-Jaroodi et al. 2010a; Al-Jaroodi & Al-Dhaheri 2011).

Service Oriented Computing (SOC) is a cross-disciplinary paradigm that is based on SOA environment to oversee distributed computing (Bichler and Lin 2006). In SOC, SOM is used to ease the design, development, and deployment of web services as well as to coordinate interaction among the components of SOA. Furthermore, SOM provides rich features such as runtime support to deploy/discover services, service transparency to clients, abstraction of the underlying environment, interoperability of devices, and integrated support for security. To fully utilize SOM within the business environment, vendors started to develop SOM functionalities that were suited to their particular business requirements. Several SOM models that were studied in Al-Jaroodi and Al-Dhaheri (2011) and Al-Jaroodi and Mohamed (2012) operate in SOC environments, yet they do not apply full security solutions. Such models only incorporate the set of functionalities required within the application domain. Moreover, with wide adoption of SOC, applications are no longer contained within a tightly controlled environment. This online exchange of information generates the risk of malicious attacks (Lazarevic et al. 2005), where an attacker crafts XML messages (SOAP request) with large payloads, recursive content, malicious external entities, or excessive nesting that causes DoS attacks. Usually in XML Denial of Service (DoS) attacks, the operational parameters of messages coming from legitimate users are changed in real-time by adding additional elements or replacing existing elements within the message. Accordingly, it is important to revise Web-service security countermeasures as Web services risks are increasing due to the open nature of communication and easy access to data (Jensen et al. 2007; Jensen et al. 2009). Presently, the available countermeasures that provide effective protection against the aforementioned types of attacks are (i) XML message validation and (ii) XML message hardening. Ultimately, one of the main features of the SOM was to provide security to SOC, which faces problems of insecure communication and configuration, information leakage and insufficient authentication (Al-Jaroodi & Mohamed 2012; Al-Jaroodi et al. 2010b). Due to the absence of standardized security guidelines (Al-Jaroodi & Al-Dhaheri 2011) several SOM approaches implement security features that are tailored to particular needs, thus hindering interoperability and reusability. Al-Jaroodi et al. (2010b) showed the need for security requirements for SOM. According to Al-Jaroodi & Al-Dhaheri (2011), the authors have proposed to develop a general set of security requirements through independent "security as a service" components. These security services can offer a variety of security functionalities that could be adapted to SOM.

The main contributions of this paper are (i) to present an application-level Bio-inspired Anomaly Detection Framework (BADF) that draws on the ideology of the Danger Theory (DT) previously proposed in (Hashim et al. 2010) for heterogeneous networks. The presented framework is designed as a generic framework that improves the security features of the SOM by applying the DT principles to protect web-service based-applications from Denial of Service (DoS) attacks. (ii) Based on BADF, we derive an architecture for a generic "security as a service" (SECaaS) web service. Our derived security service is identified as a message-protection service as mentioned in (Al-Jaroodi & Al-Dhaheri 2011). It aims to protect incoming SOAP messages against XML Denial of Service (DoS) attacks. BADF is evaluated by developing a prototype for the "security as a service" (SECaaS) architecture, and showing the ability of the SECaaS web-service to detect different types of DoS attacks induced within SOAP requests.

The rest of this paper is organized as follows; section 2 overviews related work with respect to SOAP message attacks and possible mitigation methods. Section 3 presents our Bio-inspired Anomaly Detection Framework (BADF) and the SECaaS architecture. In section 4 we describe our evaluation environment and results. Finally, in section 5 we conclude the paper and mention our future work.

2 RELATED WORK

In recent years Web service attacks have gained considerable attention from the research community. According to Jenson et al. (2007) attacks can be categorized as XML attacks, Semantic WS attacks, Cryptography based attacks, or SOAP based attacks. Vipul et al. (2011a) classified SOAP based attacks as XML injection, XSS injection, and HTTP header manipulation. All aforementioned SOAP based attacks exploit XML based messages and parsers,

and pave the way to introduce DoS attacks. DoS attacks, prevent legitimate users from accessing the attacked services, thus reducing the system's availability. DoS attacks are further categorized as Protocol Deviation Attacks or Resource Exhaustion attacks (Schafer 2014). Protocol Deviation Attacks aim to exploit the underlying protocol to make it deviate from its correct behaviour, while Resource Exhaustion attacks aim to consume the system resources. Both attacks show a destructive impact on service availability. Several papers addressed the topic of DoS attacks on Web services as it became crucial to understand the DoS impact on the operation of Web Services.

Gruschka and Luttenberger (2006) have studied two SOAP based attacks, namely Coercive parsing and Oversize payload. Coercive parsing includes recursive calls for XML tags, whereas Oversize Payload includes extremely large XML documents within SOAP messages. Gruschka and Luttenberger (2006) proposed a Check Way Gateway, which is a Web Service Firewall to validate the incoming Clients' SOAP requests through event-based parsing using a SAX (Simple API for XML) interface. The firewall generates a strict Schema from the WSDL file associated with the Web Service to validate the incoming SOAP request. They evaluated the processing and response times of the firewall validator and noted that both were within acceptable limits with respect to other intrusion detection techniques. Again Gruschka and Iacono (2009) studied XML wrapping attacks in the context of the vulnerability reported by Amazon EC2. The outcome of this study was a Security Policy Validation mechanism, which represents a practical guideline for SOAP message security validation. However, the evaluation of the proposed security validation techniques was missing. Gupta and Thilagam (2013) surveyed several SOAP based attacks out of which XML injection and Parameter tampering were reported to result in DoS. The paper discussed different attack techniques that contaminate SOAP messages to facilitate DoS attack. Among which XML injection attacks modify the XML structure of a SOAP message by inserting indefinite XML tags. Whereas, Parameter tampering attack attempts to bypass the input validation in order to access the unauthorized information to achieve DoS attack.

The authors in (Jensen et al. 2007; Jensen et al. 2009) classified the SOAPAction spoofing and oversize payload attack as SOAP based attacks, where the attacker floods the web server with XML requests that result in a web server crash. They noted that the new technologies and standards, in spite of advancing web service operation, have generated loopholes to promote DoS attacks.

The two most important countermeasures proposed in the literature and presently used to mitigate DoS attacks are XML Schema Validation (Vipul et al. 2011a), XML Schema Hardening (Vipul et al. 2011a) and Self-adaptive Schema Hardening (Vipul et al. 2011b). XML Schema Validation restricts malicious content within an XML document to make it abide by the specification of the XML Schema derived from the WSDL document. However, applying validation alone is not sufficient, as the attacker can elegantly contrive an attack by exploiting the pitfalls within the WSDL files. In this case, XML Schema Hardening should be applied as it strictly prohibits malicious content that is not contained in the XML Schema. Therefore, it is important that XML Schema should adapt to strict validation rules though schema hardening. In (Jensen et al. 2009) the authors surveyed and proposed Schema Validation, Strict WS-Security Policy Enforcement, Schema Hardening, and Event-based SOAP message processing as a countermeasure for web service attacks. Jenson et al. (2011) have studied the WS-* Specification in light of XML Signature and tried to show that XML Schema validation with a hardened XML Schema could fend XML Signature Wrapping attack. Some improvisation of XML Schema definitions is proposed to strengthen XML Schema validation. However, XML Schema hardening shows performance degradation due to increased processing time. Moreover, the proposed prevention mechanism is more specific to XML Signature Wrapping rather than Denial-of-Service attacks. Vipul et al. (2011a) proposed a new self-adaptive schema-hardening algorithm to obtain fine-tuned schema that can be used to validate SOAP messages. The proposed solution detects the Web Service attacks when compared to the other mitigation techniques. However, the comparative mitigation techniques were not clearly identified and the results presented only indicated whether the attacks were detected or not. Vipul et al. (2011b) proposed an enhanced self-adaptive schema-hardening algorithm. The presented algorithm automates schema-hardening process, and it is expected to increase the efficiency of the validation process to detect attacks. However, no evaluation results were presented for the proposed self-adaptive schema-hardening algorithm.

DoS attack is a popular form of attack in computer networks and has been studied extensively. In order to cope with DoS attacks in a heterogeneous environment, researchers have adopted ideas from the field of Biology. Hashim et al. (2010) adopted the ideology of the Danger Theory (DT) to propose an Anomaly Detection

Framework that detects DDoS attacks in heterogeneous networks. This framework detects DoS attacks through three main processes, namely Initiation Process (IP), Recognition Process (RP), and Co-stimulation Process (CP). In an internetworking environment, these three processes are triggered whenever network traffic exhibits abnormal behaviour during its operation. Abrupt changes in traffic behaviour are flagged as irregular and are identified as intrusions. Hence, the IP studies the abnormal network traffic deviation and signals the presence of malicious bandwidth attacks (such as DoS, DDoS or Worms) to RP. On its turn, the RP detects malicious anomalies in the network deviated traffic and informs nearby nodes about the possible presence of an attack. Finally, the CP adds an additional security measure by cross-examining information gained from IP and RP. CP confirms that the identified attack is really malicious or genuine and alerts the nearby nodes in the network about the presence of DoS attacks. To evaluate their framework, the authors performed different sets of DoS/DDoS and Worms attacks on an anomaly detection process, which is handled by different network domains. Analysing attack detection time and the Quality of Service (QoS) performance showed that this framework facilitates robust and adaptive anomaly detection in a heterogeneous network.

3 PROPOSED WORK

The traditional attack mitigation techniques offer solutions that strictly abide within tight, monolithic security middleware environment. As services are no longer contained within a tightly controlled environment, security solutions need to offer independent security functions. To protect against SOAP based DoS attacks, it is crucial to design and implement a flexible and secure SOAP message security validation scheme. Thus the main contributions of this paper are to (i) present an application-level Bio-inspired Anomaly Detection Framework (BADF) that uses DT principles to protect web-service based-applications against Denial of Service (DoS) attacks, then (ii) derive an architecture that is based on BADF, which will be modelled as a web service and provide "SECurity as a Service" functionalities (SECaaS) to web-service based-applications at the SOM. Our proposed system would be based on SOA architecture, where Web services communicate with three elements (i) the Service Client, (ii) the UDDI Registry, and (iii)

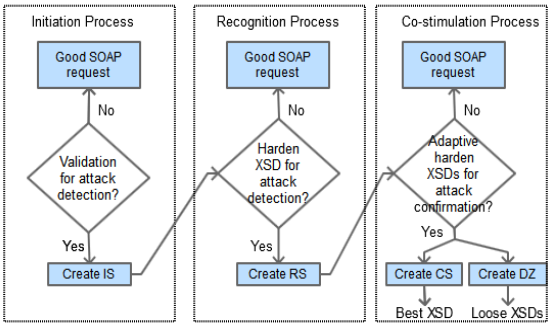

Figure 1: Biologically Inspired Anomaly Detection Framework (BADF).

the Service Provider. Our architecture will use a reformed version of the self-adaptive schema-hardening algorithm proposed by Vipul et al. (2011b) to mitigate SOAP based DoS attacks. Our choice to focus on SOAP as the communication protocol stems from the fact that most Web services are offered over HTTP using SOAP within SOA (OASIS 2015). In order to use a web service, Clients send SOAP-requests (XML document) to request a Web service, which has been previously published to the UDDI registry by the Service provider. When receiving the SOAP-request message, Service providers respond with a SOAP-response message to fulfil the Client's request. To guard against SOAP DoS attacks, the SOAP-request message need to be handled carefully before it is parsed for in-memory representation in case attacks are infused within the request. Our security service is designed to handle SOAP-request message attack and provide mitigation against XML SOAP-based attacks. In section 3.1, we first present our proposed framework and then in section 3.2 we outline the components of our derived architecture.

3.1 Biologically Inspired Anomaly Detection Framework (BADF)

Our proposed Biologically Inspired Anomaly Detection Framework (BADF) draws on the biologically inspired Anomaly Detection Framework presented in networks by Hashim et al. (2010). Our framework employs the three processes defined in Hashim et al. (2010) namely: (i) the Initiation Process (IP), (ii) the Recognition Process (RP) and (iii) the Co-stimulation Process (CP). Figure 1 shows the interaction of the three processes within BADF and the details of their operation is presented below.

- **The Initiation Process**

Receiving a SOAP request at the UDDI registry activates the Initiation Process (IP). The IP is responsible for validating the XML schema for the incoming SOAP-request messages. The XML schema validation is an important measure for checking the syntactical correctness of incoming messages. Schema validation checks for the presence of broken attributes or additional unusual elements within the message body. Detection of malfunctioned elements is considered as traces of attacks. These attack traces are marked within the XML schema and will be referred to here on as "attack vectors". In order to perform SOAP validation, the IP checks the received message structure against the XML Schema Document (XSD) associated with the corresponding Web service. Usually, the XSD is a modified XML Schema derived from the WSDL, which is a Web Service interface description document. Initially, the XSD provided by the Service Provider (SP) will be used as the reference schema for the validation step at IP. However, this Service Provider Reference Schema (SPRS) will be later replaced by the XSD updated at the RP and CP. Thus, the main task of the IP is to ensure the correctness of SOAP input parameters and operations, as specified in the web service description and as required by the Service provider. During validation, if the message does not abide by the schema structure of the XSD, the message is identified as an attack. Accordingly, the IP would send a SOAP-response message to the client indicating the presence of an attack. Since an attack can also be due to the weak strictness of the service schema document (XSD), it is important to further investigate the schema itself. The schema used for validation should be as strict as possible to hamper modification inside the message body. To combat false positive situations due to schema inefficiencies, any detected attack by the IP is sent to the RP and the CP for further investigation. Thus, In case the validation of a SOAP message fails, the IP generates a danger signal, namely the Initiation Signal (IS), for any malformed SOAP message to initiate the Recognition Process (RP). In addition, IP marks the attack vectors identified in the defective XSD in the XSD repositories to be further investigated by the RP.

- **The Recognition Process**

RP is initiated when receiving the IS signal. The RP is responsible for XML schema hardening of any defective XSDs. Thus, the RP reads the attack vectors of the message that was previously identified as an attack, as well as the corresponding XSD for further investigation. To develop a hardened XSD, the RP would first read the Web Service description document of the attacked web service. Basically, this description describes the grammar that XSD should follow. This description is parsed to develop a stricter grammar structure that would enhance the XSD operation. The structure contents, elements, rules, and definitions of the updated schema should all abide by the set of Web service description specifications as defined by the service provider and written in the WSDL. The RP generates the schema structure that represents the hardened XSD and stores the updated XSD into the Schema repository to be subsequently used by the IP in the validation step. However, the RP would not update the reference schema (SPRS) initially given by the service provider. From now on the newly hardened schema (XSD) would be used as the reference schema instead of the SPRS. After performing the hardening step, the RP decides whether the IS was issued as a result of an attack, or as a result of a lenient schema. In case of the former, the RP immediately issues the Recognition Signal (RS) for investigating the attack further at the Co-stimulation Process (CP). In case of the latter, the RP issues the Recognition Signal (RS) only after the number of logged XSDs for a specific web service has exceeded a preset threshold indicating a recurring incidences of false positive alerts.

- **The Co-stimulation Process**

The Co-stimulation Process is initiated as a result of the RS signal issued by the RP. The Co-stimulation Process is responsible for self-adaptive schema hardening for all defective XSDs that have been accumulated for all SOAP messages requesting a specific Web service. The purpose of the Co-stimulation Process (CP) is to develop improvised solutions learned from the detection of an attack at the IP and its consequent mitigation at RP. The CP is a crucial step as the validation and subsequent hardening of the XML schemata that were previously categorized as potential attacks could still possess some inaccuracies. This inaccuracy could be due to the flexible and permissive nature of the schemata, where security issues might arise, yet are not evident at first sight. Hence, a complete refinement is necessary to generate a strict XML Schema that would be learned from multiple logged-in hardened-XSD attacks for a particular Web service. Upon receiving the RS, the CP would be activated to perform self-adaptive schema hardening to develop a strict XML schema (XSD) to be later

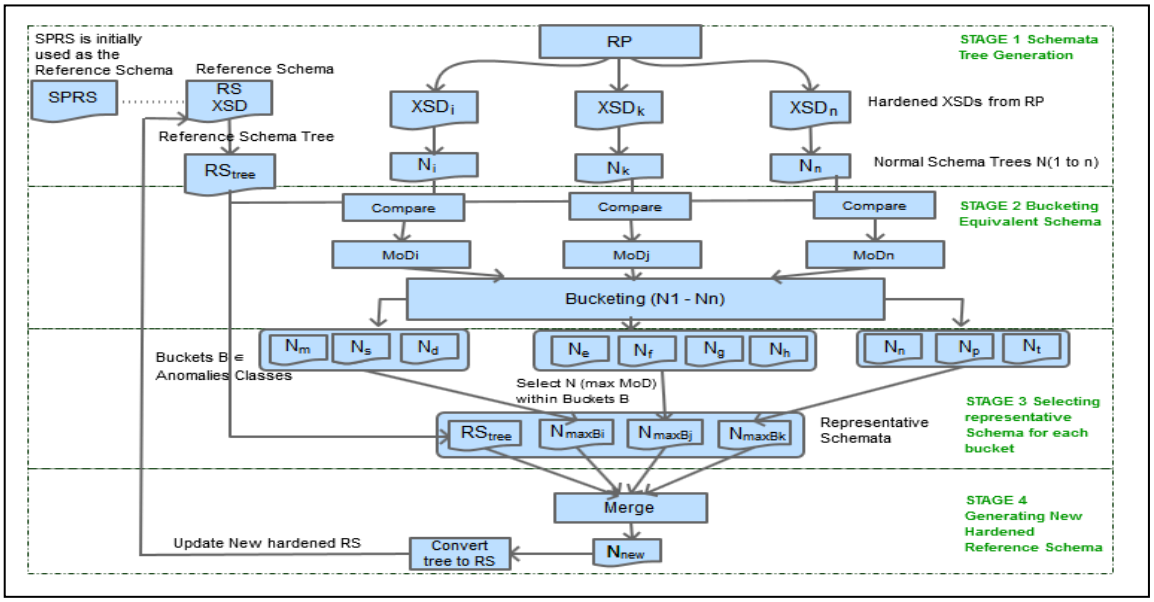

Figure 2: CP Schema hardening algorithm.

used at the IP for validation.

The self-adaptive algorithm that is used by the CP in our proposed framework is based on the self-adaptive algorithm presented by Vipul et al. (2011b) with some modifications, which will be pointed out later. Our self-adaptive hardening algorithm takes as an input all hardened XSDs that were logged by the RP to the schema repository, and produces one single hardened schema. This hardened schema will incorporate all refinements and schema hardening that were previously imposed by RP on a given reference schema as a result of multiple malformed SOAP-requests and/or attacks. This hardened schema will be used later on as the new reference schema in the validation step at the IP. The stages of our algorithm are shown in Figure 2. As shown in the figure, our algorithm is similar to Vipul in the first and second stage only. Furthermore, our algorithm takes as input the XSDs hardened by the RP in contrast to Vipul et al. (2011b) algorithm that works on XSDs generated from SOAP-requests. The stages of our algorithm are detailed below.

Stage 1 Schema Tree Generation

As pointed out by Vipul et al. (2011b), it is inconvenient to directly compare XSDs. Therefore, all hardened XSDs generated at the RP, as well as the reference XSD, will be first transformed to normalized tree representations. To generate the normalized schema tree representations we use the same methodology adopted in Vipul et al. (2011b), where each XSD is traversed and for each XSD tag encountered a node will be introduced. Each node in the generated schema tree will have the following four attributes:

- Node Name: this represents the name given to an Element and/or an Attribute. However, for nodes that do not have a name, such as nodes that represent meta-data as complexType, simpleType, sequence, etc…, Node Name will be the name for the meta-data that the node represents,
- Node Type: this represents the type of the node such as an element, an attribute, an extension, a restriction, a sequence, a complexType or a simpleType, etc…,
- Data Type: data type of a node, and
- Cardinality: this refers to the minimum and maximum number of occurrences of an element.

For each of the generated trees, a tree signature is devised using a key generation function that uniquely identifies each schema tree.

Stage 2 Bucketing Equivalent Schemas

The main aim of this stage is to determine the equivalence among the generated trees in an attempt to cluster equivalent schemata together. We opine that XSDs generated due to similar attack attempts, or similar malformed SOAP-requests will have high equivalence, thus will fall into the same bucket. As in Vipul et al. (2011b), equivalence among two schema trees is determined through the measure of difference (MoD), which is a scalar value that

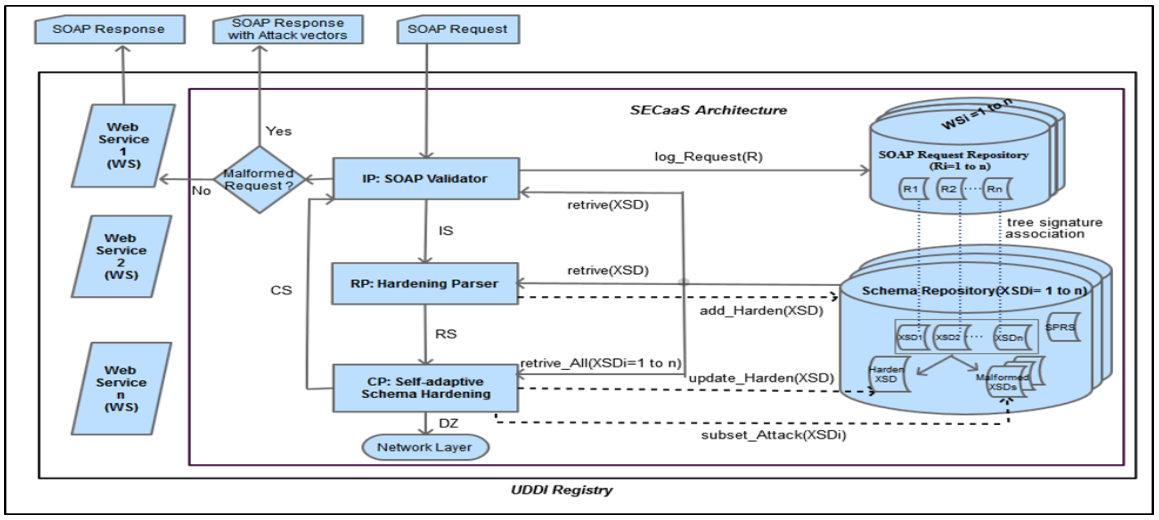

Figure 3: SECaaS Architecture.

represents an extent to which two schemas differ. For calculating the MoD among all schemata we adopt the algorithm presented in Vipul et al. (2011b), which reduces the number of comparisons by determining the equivalence among schemata by calculating the MoD between each of the schema trees and the reference schema tree. Schema trees that have similar MoD with respect to the reference schema tree will be considered equivalent and will be grouped together in the same Bucket. In this work, similar MoD means that the scalar value calculated for the MoD is within a given range. For our purpose, we create a list for each bucket that stores the schema tree signature and its equivalent MoD value with respect to the reference schema.

Stage 3 Selecting representative Schema Tree for each bucket

We speculate that the schema trees grouped into the same bucket are those that represent the XSDs that have been generated in response to similar anomaly situations. Accordingly, the schema trees grouped within the same bucket represent the XSDs that incorporate the refinements/updates that detect one class of anomalies. Furthermore, we reason that within each bucket, the schema tree with the maximum MoD with respect to the reference schema tree can be considered the representative schema tree for the rest of the trees within the same bucket as a larger MoD implies a greater degree of hardening has been applied to the schema. Thus to get a representative schema tree for each bucket, we sort the list created for each bucket in the previous step in descending order based on the value of the MoD, and pick the schema tree with the highest MoD.

Stage 4 Generating a New Hardened Reference Schema

A new hardened reference schema tree will be created by merging the reference schema tree with all representative schema trees identified in the previous step. This merging step ensures that the newly generated reference schema tree will incorporate all hardening refinements/updates required to detect the different anomaly classes that have been encountered so far. The newly generated reference schema tree will be converted back to an XSD representation and passed back to the IP to be used in the validation step. To do so, the CP activates the Co-stimulation signal (CS) to update the reference schema to the newly generated XSD. In addition, the CP activates the Danger Zone (DZ) signal that maps the left out schema trees within each bucket to associated SOAP-requests, to be used at the network level to identify sources of anomalies.

3.2 Security as a Service based on BADF

Figure 3 presents the components of the SECaaS architecture derived based on BADF. The components of the SECaaS architecture are (i) the SOAP message validator to validate SOAP messages, (ii) the Schema repository to store hardened XSDs, (iii) the SOAP-request repository to store malicious requests, (iv) the Schema hardening mitigation parser to develop hardened schema, and (v) the Reference Schema. The proposed Security

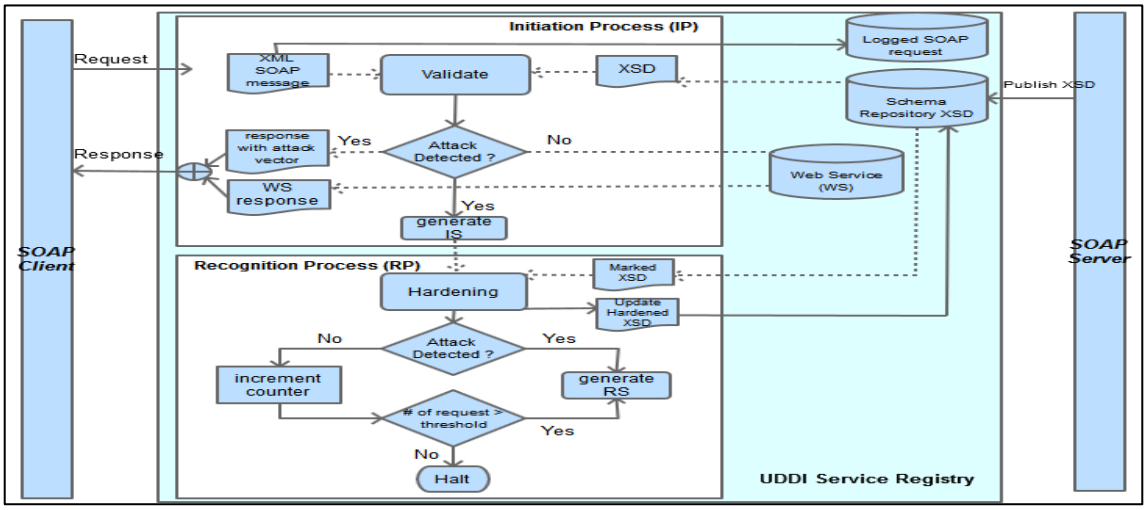

Figure 4: Flowchart for SECaaS Architecture.

Service is published within the UDDI registry as a stand-alone web service and acts as a generic Web service that secures other web services running on UDDI registry at the application layer. Each of the BADF processes operates independently and their execution is loop-free and sequential. In addition, each process depends on the different instances of the XSDs that is stored progressively in the schema repository for the correct overall operation of the security service.

Our Security Service is activated when a client sends a SOAP-request to the registry requesting a particular web service. The SOAP-request for the requested web service ("Web service 1" in Figure 3) is handed to our Security Service. Accordingly, the IP loads the XSD that is associated with the requested web service. Initially, this XSD is provided by the service provider and represents the reference schema that will be used by the message validator to validate the SOAP-request message. If the validation of the message fails, the following steps are executed (i) The IP logs the SOAP message as a malicious message in the SOAP-request repository; (ii) The IP replies to the client with a SOAP-response message identifying the attack vectors; (iii) The IP marks attack traces within the XSD; (iv) The IP triggers the IS to activate the RP.

Upon receiving the IS, the RP retrieves the XSD that has been previously accessed and marked by IP from the schema repository. The RP generates a hardened XSD by applying the hardening rules to the accessed XSD. The refined XSD is stored in the schema repository to replace the older XSD that was previously associated with the message of web service at IP. Typically, as mentioned earlier, the RP generates the Recognition Signal (RS) based on the threshold value for false positive or in response to a detected attack, to initiate the CP.

On receiving the RS, the CP is activated, which means that the dubious SOAP-requests for a particular Web service has either exceeded a given threshold value or has been identified with high confidence as an attack. The CP retrieves all accumulated XSDs for all logged SOAP-requests for a given web service from the repository, and initiates the self-adaptive schema-hardening algorithm over all XSDs for the same Web service request. The hardened XSD would update the last logged XSD by RP that was used as the reference schema in the previous processes.

4 EVALUATION

The BADF evaluation will be performed by evaluating the SECaaS architecture that we have derived in the previous section. We have developed a prototype for the SCEaaS architecture on a localhost and tested its behaviour against DoS SOAP-attacks. The SECaaS prototype, as shown in Figure 4, is composed of the main components of the SOA, namely the SOAP Service Client, the Service Registry UDDI, and the SOAP Server. The prototype components implement the IP and RP only, whereas the CP is not implemented, since the adaptive-schema hardening algorithm comprises our future work.

Table 1: DoS SOAP-based attacks detected comparison.

# Attack type	Attack code	Attack details	SOAP REQ & Parameters	Attack Results	
				SECaaS on	SECaaS off
Parameter Tampering	PT-1	Buffer Overflow of String Types	<name>‘Cornary temperedtext-CAD’ </name>	detected	detected
	PT-2	Buffer Overflow of Integer Types	<ldl> -1 </ldl>	detected	undetected
	PT-3	Field Manipulation causes URL manipulation	<heart_diseases RegistrationNumber="1295857444"> ……</heart_diseases>	detected	undetected
XDoS	XD-1	XML Extra Long Names	<name> Ischemic Heart Diseases </name> *repeated 50 times*	detected	detected
	XD-2	XML Namespace Prefix Attack	<xs:heart_diseases description = “Common type” -----*repeated 100 times*------>	detected	undetected
XML Injection	XIJ-1	Reference Entity Attack	<name>&xxe; </name>	detected	undetected
	XIJ-2	Internal Entity Attack 1	<name>> </name>	detected	undetected
	XIJ-3	Internal Entity Attack 2	<name><</name>	detected	undetected
	XIJ-4	Invalid XML meta-characters (quotes)	<xs:attribute heart_diseases =‘’>345675453’ </xs:attribute>	detected	detected
	XIJ-5	Invalid XML meta-characters (comment tag)	<xs:attribute treatment_cost = ‘’> 465<!--</xs:attribute>	detected	undetected
Recursive payload	RP-1	Tag recursive calls	<level> <level>--- Beginner---- </level></level>> *called 100 times*	detected	detected
	RP-2	XML Recursive Entity Expansion	<!ENTITY x8 “&x7;&x7;”> *called as* <attack>&x8;</attack>	detected	undetected

4.1 The Development Environment

In our implementation, we have used Eclipse JAX-WS as the SOAP engine and Apache Tomcat juddi-tomcat-3.3.2 on Microsoft Windows 7 as the SOAP server. For the Service Registry UDDI, we used jUDDI version juddi-distro-3.3.2. All the user and service information about the published schema were stored in the MySQL Community Server 5.5.14 database for jUDDI. All the Web services were developed using Java (Java SE) version jdk1. 6.0_21. For testing our security service against incoming SOAP messages we used SoapUI, which is a GUI for unit and load testing of SOAP web services. To evaluate SECaaS performance, its responses were compared to the responses generated by an external validation/parser tool (Mantid 2016) that is used in our base case scenario. We reverted to this evaluation methodology since the comparative evaluation with the literature mitigation techniques was not possible. This can be attributed to the fact that to the extent of our knowledge the concept of SECaaS specific to DoS attacks has not been presented before. Most of the reported work falls short of evaluation results and address general web service attacks. The techniques discussed in Gruschka and Luttenberger (2006) and Jensen et al. (2011) focused on performance evaluation and used large data sets for testing a number of web service elements. However, in our case, we care to evaluate the efficacy of SECaaS to detect SOAP based DoS attacks within the SOA. Therefore, we reverted to a local host implementation since an online implementation was not feasible.

To validate the efficacy of our security service on localhost, we built Disease Information Web services, which has multiple APIs. The service operates on 1000 different health trace dataset gathered into a disease database.

4.2 The Types of Attacks

Our evaluation scenarios use four different DoS SOAP-attacks, namely Parameter Tampering, XDoS, XML Injection, and Oversize/ Recursive payload. The Parameter Tampering attack infuses

malicious content with node/ tags within message query to deceive the validator. The XDoS attack tries to exhaust the system resources on the server by iteratively declaring strings. The XML Injection injects additional nodes or modifies existing nodes so as to change the operation parameters of the message. Finally, the Recursive payload adds additional nodes repeatedly, which are excessively large, to deplete CPU cycles. All these attacks (listed in Table 1) were performed using a malicious insider or man-in-the-middle techniques of DoS. Hence, to generate a good set of inputs to be used in testing, we looked at the WSDL document of Disease Web Service to find loopholes. These attacks traces were included in the SOAP request to measure the efficiency of SECaaS Architecture.

4.3 Evaluating the SECaaS Architecture

Our evaluation is composed of two scenarios; a base scenario and a SECaaS scenario. In the base scenario, the security service was turned off and an external parser was used to validate the SOAP requests. In the SECaaS scenario, the security service was turned on and the SOAP requests were validated at IP. In each scenario, we send one malformed SOAP-request followed by several legitimate SOAP requests. Each malformed SOAP-request comprises an attack and invokes a specific web service with some tampered input parameter. Table 1 details the different attacks that were administered to the system in each scenario and the response in each case. Some XSD hardening results are presented for two of the administered attacks, namely Parameter Tempering (PT-2) and XML Namespace Prefix Attack (XD-2) for IP and RP.

4.3.1 Buffer Overflow of Integer Type

To generate the attack PT-2, we analyzed the disease service WSDL document. From the document we were able to identify the rule for the element “ldl”. Element “Id1” accepts a 4-bit integer variable as an input and has a value that ranges between 1 and 15. The XSD declaration imposes the rule <xs:minLength value="1"/> <xs:maxLength value="15"/> without a restriction. Hence, we tried to input a string which is less than 1 or greater than 15 and checked the server response. In Figure 5, an invalid input value of -1 is passed to the element name ldl. If the input value is undetected it would

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope>
  <soapenv:Header>
  </soapenv:Header>
  <soapenv:Body>
    <addDiseaseInfo xmlns="http://localhost:8080/Heart_dise
    <heart_diseases RegistrationNumber="1295857444">
    <name>XXX</name>
    <description> atherosclerosis </description>
    <level>Beginner</level>
    <symptoms> WEIGHT GAIN</symptoms>
    <hdl> 47</hdl>
    <ldl> -1 </ldl>
    <defect_type>Complex cyanotic </defect_type>
    <medicines>statins, Niacin and fibrates</medicines>
    <treatment>surgery</treatment>
    </heart_diseases>
    </addDiseaseInfo>
  </soapenv:Body>
</soapenv:Envelope>
```

Figure 5: PT-2 SOAP Request.

```
<soapenv:Body>
  <addDiseaseInfo xmlns="http://localhost:8080/Heart_disease
    <faultcode>soapenv:Server:ServerException</faultcode>
     <faultstring>Invalid input at line 12 </faultstring>
  </addDiseaseInfo>
</soapenv:Body>
```

Figure 6: SOAP Response at SECaaS on case.

exploit the parser to result in a buffer overflow exception. In the base case, when the SECaaS was turned off the invalid input was not detected. However, when the SECaaS was turned on, the invalid input resulted in a server exception to handle a buffer overflow attack situation. The result of attack detection is shown in Figure 6. For hardening, the RP imposed a restriction as a type of “integer” over all declared tags in the older XSD as new hardened XSD. This restriction requires the input to be of type integer for all declared tags.

```
    Old XSD: < xs:minLength
value="1" xs:maxLength value="15">


    New XSD: < xs:restriction
base="xs:integer"  xs:minLength
value="1" xs:maxLength value="15">
```

4.3.2 XML Namespace Prefix Attack

The purpose of this attack is to place as many attributes in an element, so that a buffer overflow would occur before the namespace prefix gets declared. Once a namespace and its prefix are declared, adding the prefix label to elements and attributes qualifies the entity for the namespace. Placing the target namespace attribute at the top of

Figure 7: XD-2 SOAP Request.

Figure 8: SOAP Response at SECaaS on case.

the XSD schema means that all entities defined in it are part of this namespace. In this attack, we enforce a buffer overflow by infusing a large number of attributes in an element. Thus, before the parser could reach the namespace prefix declaration to generate an error, a buffer overflow occurs. Hence, we tried to play with the element tag <description>, to induce our attack. As shown in Figure 7, within the SOAP request, the attribute description is inserted iteratively to overflow the parser. In the base case, the attack was detected only when the attribute is iterated for a maximum of 5 times. Otherwise the parser crashed. However, when the SECaaS was turned on, the IP detected the attack even when we continually increased the attribute iteration. Figure 8 shows that server throws an exception for the detected attack. The RP placed some restrictions over the prefix declaration in XSD. The description element within schema is limited to an occurrence of 5 times, to avoid its iteration. Moreover, the input string is constrained by the maximum length to 100. The element name "description" in the older XSD is replaced with the restriction in the new XSD as shown in XML code.

```
Old XSD:
<xs:simpleType name="description"
xs:minOccurs="1" maxOccurs="5"
minLength value="15"maxLength
value="100”>

New XSD:
<xs:simpleType name="description"
  xs:minOccurs="1" maxOccurs="5" >
  <xs:restriction base="xs:string"
  minLength value="15" maxLength
  value="100”>
  </xs:restriction>
</xs:simpleType>
```

Table 2: Attack detection results (%).

# Attack type	# Attack code Mix	% Attack Results	
		SECaaS on	SECaaS off
Parameter Tampering	PT-1, PT-2, & PT-3	63.63	36.36
XDoS	XD-1, XD-2, & XIJ-1	70	50
XML Injection	XIJ-1, XIJ-2, XIJ-3, XIJ-4, & XIJ-5	80	50
Recursive payload	RP-1 & RP-2	80	40

As shown in Table 1, SECaaS prototype was capable of detecting all contrived attacks in contrast to the base case, which missed several attacks. The summary of the results of our evaluation can be found in Table 2. As can be seen, the SECaaS prototype is more capable of attacks detection compared to the base case. We note that our evaluation has covered only a subset of the possible SOAP attacks mentioned in literature, since the exhaustive enumeration of all possible SOAP attacks is not possible. However, we claim that the results obtained provide a proof of concept and show the efficacy of our SECaaS architecture to defend web services against SOAP based attacks.

5 CONCLUSIONS

In this paper, we have presented a generic application-level Bio-inspired Anomaly Detection Framework (BADF) that improves the security features of the SOM by defending web-service based-applications from Denial of Service (DoS) attacks. Based on BADF we have derived a

"Security as a Service" (SECaaS) architecture that employs SOAP message validation and subsequent schema hardening to defend against DoS attacks. To evaluate our work we have developed a prototype for the (SECaaS) architecture and tested it against several DoS SOAP-based attacks. Results show that our prototype was capable of detecting all attacks administered to the system. Our future work will focus on implementation and evaluation of CP algorithm. Although the proposed work presents the detection of some DoS attacks, a formal proof for mitigation is missing. Thus, upcoming work would focus on performance evaluation of the presented work in comparison to other techniques.

REFERENCES

Al-Jaroodi, J., & Al-Dhaheri, A., 2011, 'Security Issues of Service-Oriented Middleware' *in International Journal of Computer Science and Network Security*, vol. 11, no.1.

Al-Jaroodi, J., Mohamed, N., & Aziz, J., 2010a, 'Service Oriented Middleware: Trends and Challenges' in *Proceedings of the 2010 Seventh International Conference on Information Technology: New Generations (ITNG)* , IEEE CPS, Las Vegas, USA.

Al-Jaroodi, J., & Mohamed, N., 2012, 'Service-oriented middleware: A survey' in *Journal of Network and Computer Applications*, vol.35, p. 211–220.

Al-Jaroodi, J., Jawhar, I., Al-Dhaheri, A., Al-Abdouli, F., & Mohamed, N., 2010b, 'Security middleware approaches and issues for ubiquitous applications' in *Science Direct Computers and Mathematics with Applications*, vol. 60, p. 187–197.

Lazarevic, A., Kumar, V., & Srivastava, J., 2005, 'Intrusion Detection: Survey' in *Managing Cyber Threats: Issues, Approaches, and Challenges*, Springer Science and Business Media, Inc., New York, p. 19- 78.

Bichler, M., & Lin, K.J., 2006, 'Service-Oriented Computing' in *IEEE Computer*, vol. 39, no. 3, p. 99–101.

Hashim, F., Munasinghe, K.S., & Jamalipour, A., 2010, 'Biologically Inspired Anomaly Detection and Security Control Frameworks for Complex Heterogeneous Networks' in *Proceedings of the IEEE Transactions on Network and Service Management*, vol. 7, no. 4, p. 268–281.

Gruschka, N., & Luttenberger, N., 2006, 'Protecting Web Services from DoS Attacks by SOAP Message Validation' in *IFIP TC-11 21st International Information Security Conference*, SEC 2006, vol. 201, p. 22–24.

Jensen, M., Gruschka, N., Herkenhoner, R., Luttenberger, N., 2007, 'SOA and Web Services: New Technologies, New Standards - New Attacks' in *ECOWS'07 Fifth European Conference on Web Services*, p. 35-44.

Jensen, M., Gruschka, N., & Herkenh, R., 2009, 'A Survey of Attacks on Web Services' in *Journal Computer Science - Research and Development*.

Jensen, M., Meyer, C., Somorovsky, J., & Schwenk, J., 2011, 'On the Effectiveness of XML Schema Validation for Countering XML Signature Wrapping Attacks' in IEEE, viewed 12 January 2015, from http://dx.doi.org/10.1109/IWSSCloud.2011.6049019.

Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, JJ., Nielsen, HF., Karmarkar, A., & Lafon, Y., 2007, 'SOAP Version 1.2.' in *W3C Recommendation specification*—SOAP Version 1.2, vol. 24.

"Web Services Security: SOAP Messages Security 1.1", *OASIS Standard*, viewed 2 March 2015, from http://www.oasis-open.org/.

Vipul, P., Mohandas, R., & Pais, A. R., 2011a, 'Attacks On Web Services And Mitigation Schemes' in *Proceedings of the 2010 International Conference*, Security and Cryptography (SECRYPT).

Vipul, P., Mohandas, R., & Pais, A., 2011b, 'Safeguarding Web Services Using Self-Adaptive Schema Hardening Algorithm' in *Advances in Network Security and Applications, Communications in Computer and Information Science*, vol. 196, Springer Berlin Heidelberg, Chennai, India.

Gupta, A. N., & Thilagam, P. S., 2013, 'Attacks On Web Services Need To Secure Xml On Web' in *Computer Science & Engineering: An International Journal (CSEIJ)*, vol. 3, no. 5.

Gruschka, N. & Iacono, L., 2009, 'Vulnerable Cloud: SOAP Message Security Validation Revisited' in *IEEE International Conference on Web Services ICWS*.

Schäfer, G., Sisalem, D., & Kuthan, J., 2014, 'Denial of Service Attacks and Sip Infrastructure Attack Scenarios and Prevention Mechanisms', viewed 10 May 2014, from http://www.iptel.org/~dor/papers/Sisalem1204_DoS.pdf.

W3C.2008, viewed 22 March 2015, from https://www.w3.org/TR/xml.

W3C.2001, viewed 22 March 2015, from http://www.w3.org/TR/wsdl.

W3C.2007, viewed 22 March 2015, from http://www.w3.org/TR/soap/

W3C.2001 viewed 22 March 2015, from https://www.w3.org/XML/Schema.

OASIS UDDI Specification TC, viewed 22 March 2015, from https://www.oasis-open.org/committees/uddi-spec/faq.php.

MANTID *Using XML Schema*, viewed 18 Oct 2016, from http://www.mantidproject.org/Using_XML_Schema.