2011

Service Oriented Architecture (SOA) Security Models

Majd Mahmoud Al-kofahi
Iowa State University

Service Oriented Architecture (SOA) Security Models

by

Majd Mahmoud Al-kofahi

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Computer Engineering

Program of Study Committee:

Thomas E Daniels, Major Professor

Doug W Jacobson

Mani Mina

Steve F Russell

Anthony M Townsend

Iowa State University

Ames, Iowa

2011

DEDICATION

I would like to dedicate this thesis to my husband Mohammad and to my parents whom without there support, patience and love I would not have been able to complete this work. I would also like to thank my friends and family for their loving guidance and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

The work in this thesis wouldn't have been possible without the supervision, guidance, patience and support of my advisor Dr.Thomas E.Daniels who stood by me, supervised and directed me throughout this research and the writing of this thesis. I would also like to thank my committee members for their efforts and contributions to this work.

The first part of the work presented in this thesis which is the development of Service Clark Wilson Integrity Model (SCWIM) for SOA networks was sponsored by the National Science Foundation-Sponsored Center for Information Protection under award ECC0540362, and partially funded through the IBM Faculty Award.

ABSTRACT

Interest in Service Oriented Architecture (SOA) is rapidly increasing in the business world due to the many benefits it offers such as reliability, manageability, re-usability, flexibility, efficiency, and interoperability.

Many security technologies, models and systems have been developed for SOA, covering one or a combination of security aspects such as authentication, authorization, encryption, trust, confidentiality or access control. Even though many security areas have been thoroughly investigated, many are still unexplored such as integrity protection and SOA intrusion detection systems.

In this thesis we are proposing *Service Clark-Wilson Integrity Model (SCWIM)*, a top down integrity model for SOA capable of describing sufficient conditions to protect data integrity in any SOA implementation based on the original Clark-Wilson Integrity Model. Our model can form the basis for system security audits and assist SOA architects in developing systems that protect data integrity as well as providing guidance for evaluating existing SOA systems.

We are also proposing *SOA Specification Based Intrusion Detection System* capable of detecting intrusions affecting service behaviors in SOA networks. A SOA testbed was implemented, configured, and modified to accommodate the needs of our research and to work as the base for the development of our specification based IDS. We believe that our IDS will provide a low false negative/positive rate and will be able to detect known and novel attacks that affect the behavior of the monitored services.

CHAPTER 1. SOA Security Overview

1.1 Introduction

1.1.1 Background

SOA is a collection of loosely coupled and independent services (or resources) each with a well defined interface that help the service interact with other services regardless of their implementation or platform. Services are offered on demand and can range from a simple service were only one service is involved to a higher level service composed of many services. Services can be delivered to an end user, application or to another service there is no need for human intervention. Services in SOA networks are most likely implemented using web services technology. This does not mean that other types of technologies are not applicable [1]. Any technology that promote sharing, reuse, interoperability and have a mean for advertising the services is a technology that can implement SOA. In our work we use a SOA testbed implemented using web services. More details about the testbed will be discussed in chapter 3. The most popular data format and protocol in use for web services are XML and SOAP [2]. In most SOA implementations a directory system known as UDDI is used for web service discovery and publication. Web servers advertise services using a well defined interface or WSDL file and uses SOAP messages as a communication mechanism between web services.

The creation of a business processes from composite web services and the coordination between these services in the SOA environment follows one of two strategies; either an orchestration strategy or a choreography strategy. In the choregraphy strategy interactions between web services are specified from a global perspective, whereas in the orchestration strategy it is specified from a single point of view of one participant, the orchestrator. [3].

We are mainly interested in securing SOA networks that are implemented using web service technology since it is the most common implementation technology in use today. In this chapter we will try

to give a brief summary of SOA and its definition, requirements, security, security models,and attacks. We will conclude with the contributions of our work.

1.1.1.1 SOA Definition

There is no widely accepted formal definition for SOA; different groups have different definitions depending on their perspective or interest. World wide web consortium (W3C) [4] defines SOA as ”A set of components which can be invoked, and whose interface descriptions can be published and discovered”, component based development and integration (CBDI)[5] on the other hand defines SOA as ”The policies, practices, frameworks that enable application functionality to be provided and consumed as sets of services published at a granularity relevant to the service consumer that can be invoked, published and discovered, which are abstracted away from the implementation using a single, standards based form of interface”. The IBM SOA Center of Excellence [6] defines SOA as ”An enterprise-scale IT architecture for linking resources on demand. These resources are represented as business-aligned services which can participate and be composed in a value-net, enterprise, or line of business to fulfill business needs. The primary structuring element for SOA applications is a service as opposed to subsystems, systems, or components”.

1.1.1.2 SOA Requirements

Despite the differences in SOA definitions we can still conclude that SOA environments have the following basic components and characteristics [7, 8, 9]:

- **Loose coupling:** Minimize dependencies between services where a change in one service does not require changes in linked services.
- **Interoperability:** Each service should have a well defined interface that will describe the service and help SOA achieve interoperability between different services under different platforms and technologies.
- **Composability:** Any service in the SOA network should be able to be composed with other services to create a higher level service or business process.

- **Reusability:** Services should be useful for developing several applications.
- **Service Contract:** Each service interface defines a contract that manages the communication between services.
- **Encapsulation:** Services must hide all of the information related to it from the outside world except what is described in the service contract.
- **Abstraction:** Description of the service provided should be separate from the implementation.
- **Autonomy:** Services have control over the logic they encapsulate. There is no need for human intervention.
- **Discoverability:** Users or other services Exist some mechanisms for finding service providers.
- **Granularity:** This characteristic describes the extent to which the SOA network is broken down into services and sub services, so the finer the granularity is the more services and sub services the SOA network has.
- **Modularity:** It increases the extent to which SOA network is made of separate service modules that can be used interchangeably.
- **Componentization:** It states that the SOA network should be made of several separate and more manageable components or services to increase flexibility and manageability.
- **Flexibility:** The SOA network is flexible in that there is no limit on the number of services that can be part of the network, the platform used, the location of these services,language they are implemented in...etc.

1.2 Security in SOA

Security in the SOA environment involves securing all elements of the SOA network; services, messages, data stores. Due to the decentralized and distributed nature of SOA networks and the use of web services, different services are distributed across different platforms and enterprises which means that data flows in all directions and needs to be protected at all times [10]. Many users can use the available

services concurrently leading to global consistency problems if not carefully managed. In such an open environment, distinguishing legitimate service requests from illegitimate ones, securing the integrity of the messages in transit, the data and meta data stored, and the communication channel becomes a challenge. The traditional mechanisms that are available are typically based on point-to-point security, meaning that they will protect a communication between one endpoint and another endpoint. And they will enable authentication between those two endpoints but they will not provide the kind of security required to protect information when it is sitting in an intermediary or propagate that authentication information to the next stop in the process. In a point to point communication the technologies for providing integrity are well known (e.g. SSL, TLS) but for a distributed environment like SOA where the message might go through multiple intermediaries we need to think about end to end communication security instead [11].

1.2.0.3 SOA Security Models

The primary security aspects or functions required by most systems are confidentiality, integrity, and availability.

In SOA this has been done using different technologies, standards and models [6, 11, 12, 13, 14, 15]. Examples include WS-Security, WS-Trust, WS-Security Policy, WS-Federation, SAML, XACML, Trust-but-Verify, Attribute Based Access Control (ABAC) and many more. These SOA security standards provide message integrity, confidentiality, authentication, access control and trust as described below:

- *WS-Security (WSS)* is a communication protocol that provides message integrity protection, confidentiality, and proof of origin using XML Encryption and XML Signature [11]. Although this standard provide integrity it does not guarantee the integrity of the whole message. It guarantees the integrity of each part by itself.

- *WS-Trust* defines extensions to WS-Security to provide mechanisms to establish trust relationships between different clients in different security infrastructures [11].

- *WS-Security Policy* manages the specifications of security requirements and capabilities between senders and receivers [11].

- *Security Assertion Markup Language (SAML)* enables identity management, provides a standard protocol to implement SSO (Single Sign On) and provides a standard security token that can be used with the WS-Security framework [16].

- *Extensible Access Control Markup Language (XACML*) uses any available information to decide if access to resource should be permitted, and associates additional actions with the decision like destroying the data after a period of time [17].

- *Trust-But-Verify* approach separates authorization process into different on-line and offline phases [14, 12].

- *Attribute Based Access Control (ABAC)* uses a finer grained approach than other access control mechanisms such as Identity Based Access Control (IBAC), Role Based Access Control (RBAC), or Lattice Based Access Control (LBAC) [12, 15].

Integrity protection and intrusion detection are two unexplored areas in SOA security. Even though many technologies and models were developed for SOA security; none of them was entirely devoted to ensuring the overall integrity and global consistency of all services and data items in a specified SOA network or developed to detect possible intrusions on them. For example, although WSS standard provides end to end message integrity it does not protect the integrity of other types of data like meta data or code associated with each service and does not guarantee the global consistency of all data items participating in a specific SOA network.

1.2.1 SOA Attacks

SOA environment is vulnerable to various types of attacks especially if it was implemented using web service technology. Both SOA and web services are two rapidly changing and developing areas that are being used by millions of people all around the world, as a consequence they are getting more complex and open to a huge set of old and new attacks. Most of the attacks on web services and SOA are taking place at the application service layer since web services usually communicate using XML and soap messages. This XML based traffic often bypasses network defenses such as firewalls, gateways and IDS which usually rely on TCP/IP packet filtering models. Many security standards

and mechanism were proposed to secure web services and SOA networks as we discussed earlier in the previous section, but none was entirely devoted to guarantee the integrity or capable of detecting intrusions of SOA networks. Our goal in this section is to discuss some of the attacks that we belive will affect the behavior of the service and can be detected by our proposed IDS system which will be discussed in more details in 4. Many types of attacks can take place in a SOA network we are mainly concerned with the following types of attacks:

1.2.1.1 Injection Attacks

This type of attack is one of the oldest known attacks for web services. Injection attacks occurs when there is no validation performed on user input and if there is no separation between the user input and the application or program instructions [18]. Examples on this type of attacks include:

- SQL Injection: This attack is responsible for modifying, executing or spoofing database content by injecting or inserting sql commands into the user input and as a consequence affect the execution of predefined sql commands, and it affects applications or web services that support database.

- XML Injection: This type of attack is one of the simplest injection attacks where XML structure of the soap message or any other XML document is modified by inserting new parameters or values into their XML tags [19].

- XQuery Injection: XQuery language is similar to SQL language. XQuery injection attack is the XML variant of the sql injection attack and it uses unvalidated user input that is passed to XQuery commands to enumerate elements, inject command or execute queries.

- XPATH Injection: XPATH is similar to sql language and but it is responsible for querying XML document instead of the database. and fetching data from the database or supply information.If XPATH injection gets executed successfully, an attacker can bypass authentication mechanisms or cause the loss of confidential information.

We used XML injection attack to test our specification based IDS as will be discussed in more details in 4.

1.2.1.2 Schema Poisoning Attack

This type of attack is responsible for replacing, modifying or damaging the XML schema which provides the structure and content definition of XML documents.

1.2.1.3 Denial Of Service Attacks (DoS)

Although we are not interested in denial of service attacks , since most of the time it doesn't change the behavior of the attacked service but we felt the necessity of talking about it. Denial of service attacks are among the most encountered types of attacks against web services and SOA, they don't change the service or its behavior but certainly can block the use of the service. There are several kinds of denial of service attacks some of which are discussed in [19] examples include:

- Resource exhaustion.
- Coercive parsing.

These sets of attacks justified the need for a powerful intrusion detection system (IDS) that is capable of detecting known and novel attacks against web services and SOA that affect the behavior of the participating services. More details about the proposed IDS will be given in 4.

1.2.2 Research Contributions

The main contributions of the work in this thesis are:

- Providing a well founded integrity model for SOA that will provide sufficient conditions to protect data integrity, guarantee the overall consistency regardless of SOA implementation, work as an overarching integrity model for all the available SOA security models, standards and technologies and finally form the basis for system security audits and assist SOA architects in developing systems that protect data integrity as well as providing guidance for evaluating existing SOA systems. More discussion about this model will be given in 2.
- Implementing a SOA testbed and setting up the environment for the development of our specification based IDS. More details about the testbed, its configuration, setup and modification will be given in 3.

- Proposing a specification based intrusion detection system for SOA networks capable of detecting intrusions that affect the behavior of services. A set of specifications are learned about the services that characterize their behavior. We assume that all the services studied in our system are web based services that use SOAP messages to communicate with each other. The detection technique used is based on the assumption that any change in the behavior of a service is an intrusion if this behavior does not meet a set of known specifications developed for this service. More details about this IDS and its development stages is given in 4.

- Extendeding the SOA testbed to allow the monitoring of SOAP messages and the injection of attacks to test the proposed intrusion detection system. An open source soap monitoring tool called wsmonitor was modified and incorporated into the testbed to achieve these two goal. More details about these additions is given in 3.

  Our novel approach will provide the following advantages over the existing IDS system available for services:

  - Our proposed IDS does not require knowledge of the underlying service code. Which makes it easy to implement.
  - Our proposed IDS can detect all abnormal behaviors of the monitored service that might lead to an attack.
  - Our proposed model can adapt to any changes in the service implementation, and will still be effective regardless of the programing language or platform used, which makes it flexible and implementation independent.
  - We believe that our IDS will give a low false negative/positive rate.

The development of our proposed specification-based IDS will be discussed in more details in 4.

CHAPTER 2. Service Clark-Wilson Integrity Model

2.1 Introduction

A security model is an abstraction of a system that gives some security guarantees given a true mapping of the model to the system. Systems rarely map cleanly to any security models, but security models remain instructive for implementing and evaluating systems. For instance, the Bell-Lapadula Model (BLP) [20, 21] is lattice based and restricts subject's rights based on level and clearance. Systems that implement BLP had processes that violated the models requirements and required some level of trust. Despite these issues, such security models can still suggest security guidelines and motivate requirements for development.

Integrity, confidentiality and availability are all key security aspects for the success of any service business. Depending on the type of business or the transactions taking place one aspect might gain more importance than the others. For example, in an inventory control system managing who can modify the data is more important than releasing it. Most Commercial and industrial firms are more concerned with accuracy than disclosure of data [22]. On the other hand, if we are talking about a service that uses a lot of confidential data like credit card numbers or passwords then confidentiality becomes an issue. Many security models have been proposed for SOA networks as we discussed in the previous chapter but non was totally dedicated to SOA's integrity and consistency control. What makes integrity more important in SOA is the need to guarantee the trustworthiness of the data used by different services and users. In this chapter we are proposing Service Clark Wilson Integrity Model (SCWIM) [23] which is a modified version of Clark Wilson Integrity Model (CWIM).

This model is a strong candidate for our proposed integrity protection model because unlike other integrity models, it is suitable for business environment. It is based on transactions as the basic operation which represents many commercial systems,SOA is an example of such a commercial system, more

realistically than other integrity models [24]. Its structure is similar to SOA's structure, and it captures SOA's requirements and characteristics. However, despite the similarities between CWIM and SOA, CWIM can not be directly applicable to SOA.

In this section a brief summary about (CWIM) is given with more discussion to be continued in section 2 when we discuss the modified version SCWIM.

2.1.1 Clark-Wilson Integrity Model(CWIM)

The Clark Wilson Integrity Model [24] was developed in 1987 by David Clark and David Wilson. The history behind developing this model came from the need to think about integrity since earlier security models focused on confidentiality as the main security issue. This model is based on the following key notions:

- **Constrained Data Items (CDIs):** data items subject to integrity control.
- **Unconstrained Data Items (UDIs):** data items not subject to integrity control. These data items must be validated before affecting the state of any CDI.
- **Integrity Verification Procedures (IVPs):** verifies the integrity of CDIs against the integrity constraints of the system.
- **Transformation Procedures (TPs):** responsible for changing the state of the data in the system from one valid state to another valid state.

It is worth knowing that TPs implement well-formed transactions only, a well formed transaction should leave all CDIs in the system in a valid state if it starts out with a valid state. All inputs entered by users are assumed to be UDIs and they must be filtered, and updated to CDIs or rejected based on the integrity constraints associated with the IVP. A set of certified relations control all transactions and data items and makes sure that separation of duty and well formed transactions are taking place. These relations are summarized in a set of certification and enforcement rules. Figure 1 show the relationships between different entities in CWIM.

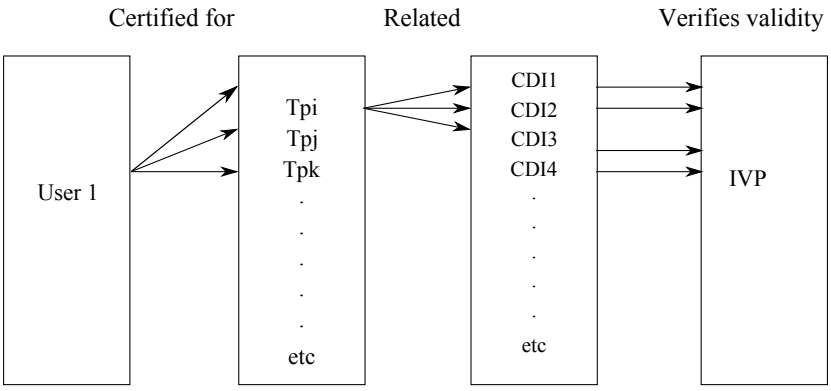

Figure 2.1 The relationship between different entities in Clark-Wilson Integrity Model

2.2 Service Clark-Wilson Integrity Model (SCWIM)

As was mentioned earlier, CWIM is a good candidate because its structure is similar to SOA's structure and it captures most of its requirements and characteristics as will be shown later in this section when we discuss (SCWIM)[23] enforcement and certification rules. However, despite the similarities between CWIM and SOA, CWIM can not be directly applicable to SOA. Therefore, we propose Service Clark-Wilson Integrity Model (SCWIM) capable of incorporating the notion of a service as an integration of sub-services, service contract, concurrency and consistency, transaction sequencing and service dependencies into the original certification and enforcement rules.

In this section we will suggest some modifications and extensions to these rules in an effort to achieve the goal of preserving data integrity.

As the previous definitions of SOA imply, SOA networks consist of a collection of services collaborating to perform a business processes. For each initiated request there is a base service that will be in charge of contacting other services whom in turn might contact more services, this process can continue until the request is fulfilled. We will refer to this base service as the *Root Service (RS)*. All data items in the network are classified into *CDIs (constrained data items*) or *UDIs (unconstrained data items)* based on how their integrity affects the overall integrity of all services and data items. Each service is responsible for a set of CDIs. And all updates and changes performed on these CDIs are done by this service.

Two different types of procedures are used to check and maintain the integrity of all services and

CDIs, these are *IVPs (Integrity Verification Procedures)* and *TPs (Transformation Procedures)*. The IVPs ensure that all CDIs in the network meet the integrity constraints of the system before the start of any transaction (i.e. the system is in a valid state). The TPs implement the functionality of the SOA and are required to move all services and CDIs from one valid state to another.

Each root service is considered to be a *Well Formed Service (WFS)* if it leaves all sub-services CDIs in a valid state after the completion of a TP and ensures the global consistency of all CDIs in all services to still be valid despite failure of/or unexpected input to any service or sub-service.

Each TP consists of a set of *Service Transformation Procedures (STPs)*. These STPs represent sub-transactions between sub-services as follows. STP_1 is formed of all transactions taking place between all sub-services from the root service to the final service. STP_2 is the set of transactions from the second called service to the end and so on. In order for the TP to move all services from one valid state to another valid state, it is necessary but not sufficient that all STPs be completed successfully and all services be left in a valid state. If all sub services are mutually in a valid state before and after a well formed service, then the system should be in a valid state after fulfilling the requested service.

Figure 2.1 is a sample SOA network that shows the relations between all of the previously defined concepts and definitions. Each service contains a set of CDIs like service S9, but to reduce the clutter in the diagram this was not shown in other services. The figure also shows how STPs are laid out inside a TP. Before the start of the TP, the IVPs validate the integrity of all services and CDIs in the diagram to make sure that the global consistency is maintained and all services and CDIs are in a valid state. We realize that in general an IVP is at best challenging to implement by some types of applications. Based on the previous definitions the following set of modified rules form the base for the Service Clark-Wilson Integrity Model (SCWIM) [23] we are proposing for SOA. These rules are classified into two different types: enforcement rules and certification rules. The enforcement rules are enforced by all applications and services that use the model, whereas the certification rules are certified by the security officer or system owner with respect to an integrity policy [24].

2.2.1 Encapsulation

Encapsulation increases the decoupling between different services and as a result increases flexibility.The following rule captures the encapsulation requirement in SOA.

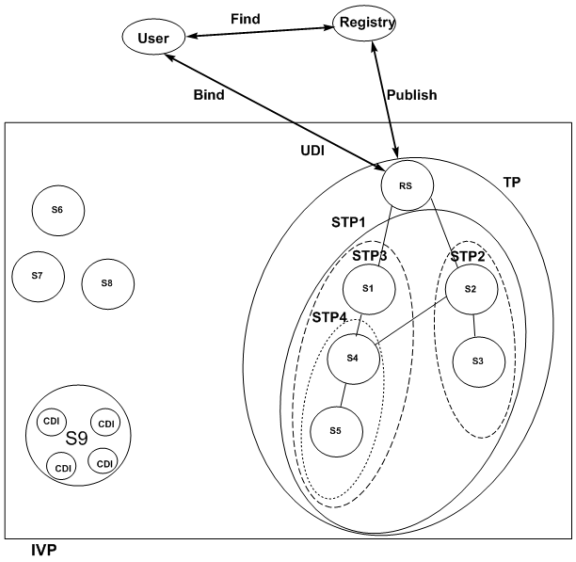

Figure 2.2 SCWIM entities interactions

Certification Rule 1:

All CDIs are associated with a service and each service is responsible for performing updates and changes to its CDIs.

2.2.2 Service Contract

Since SOA networks might be widely distributed among different networks, infrastructures and geographic locations with a diverse mix of new and old technologies, they need a well defined service contract that will maintain consistency and integrity of the data and manage the communication between different services regardless of the combination of systems or technologies involved.

Certification Rule 2:

Each service must be certified to have a well defined service contract that captures all of these certification and enforcement rules. It must be certified to maintain the consistency and integrity of the data regardless of the combinations of networks, infrastructures, locations and technologies involved.

2.2.3 Concurrency and Consistency Control

In SOA environment multiple transactions can take place at the same time between different services that are not necessarily dependent on each other. Although this concurrency can improve the

performance of independent services, it can cause problems in the case of dependent services. This can lead to consistency problems if not carefully managed by a set of concurrency and consistency rules specified for this purpose.

In the original Clark-Wilson model, concurrency was not a problem because we were dealing with one host and one copy of each CDI. When one CDI is involved in a transaction, no other transaction is able to use it until the first transaction is completed and the state of the CDI is updated. Concurrency would not be a problem, unless we are dealing with different copies of the same CDI on different machines or services, because this might affect the global consistency where some of the services might not be using the last updated version of that CDI.

It is not possible to enforce consistency constraints after each action. One may need to temporarily violate the consistency of the system state while modifying it [25]. What is important here is to maintain the global consistency of all services and CDIs once the TP is completed especially in the case of concurrency.

Certification Rule 3:

If a CDI can be used and updated by two different services S1 and S2 simultaneously. Then both services must be certified to ensure the mutual consistency of the updated CDI and all other CDIs.

Certification Rule 4:

Concurrent TPs must be certified to maintain the global consistency of all services and CDIs once they are completed.

2.2.4 Authentication and Authorization

Users and services might have several identities to use in different networks or for different services. These identities must be authenticated and subject to the same security controls, as described in the following rule.

Enforcement Rule 1:

Each service must authenticate the identity of all subjects attempting to execute a TP whether these subjects were users or services. As well as authenticating all propagations of these identities across all dependent services for this TP.

The notion of an STP needs to be incorporated in all of the relationships used in the authorization pro-

cess to make sure that each STP is certified and authorized to use certain CDIs. The subject in these relations can be identified by a userID or a serviceID and the data items can be local to the service or a reply from another service. The following two rules capture that.

Certification Rule 5:

All TPs and STPs must be certified to be valid. That is, they must take a CDI to a valid final state, given that it is in a valid state to begin with. For each TP, and each set of CDIs that it may manipulate, the security officer, must specify a ”relation”, which defines that execution. A relation is thus of the form: (TPi, STPi, (CDIa, CDIb, CDIc)), where the list of CDIs defines a particular set of arguments for which the TP has been certified.

Enforcement Rule 2:

Each service must maintain a list of relations of the form: (SubjectID, TPi, STPi, (CDIa, CDIb, CDIc)), which relates a subject, a TP, an STP and the data objects that these TPs and STPs may reference on behalf of that user. It must ensure that only executions described in one of the relations are performed.

In addition, it is necessary to ensure that all manipulations on data items are not done arbitrarily but in a constrained manner that will maintain the integrity of this data item and other data items to guarantee the global consistency of all services and sub services. The concept of well formed transactions captures that in the following rule.

Certification Rule 6:

All STPs must be certified to be part of a well formed TP. This certification rule should capture all the dependencies between services and sub services.

2.2.5 Separation of Duty

The principle of separation of duty implies that the agent responsible for creating or certifying a well formed transaction must not be allowed in the process of implementation or execution of that transaction.

Certification Rule 7:

The list of relations maintained by each service must be certified to meet the separation of duty requirements.

Enforcement Rule 3:

Only the subjects permitted to certify entities may change the list of such entities associated with other entities: specifically, those subjects associated with a TP. An agent that can certify an entity may not have any execute rights with respect to that entity.

2.2.6 Transaction Sequencing

To ensure the integrity of the business process, transactions must be performed in a specific sequence. In many cases, applications implement a first in/ first out (FIFO) queue by waiting for each transaction to be completed before the next one in the queue is processed (processing transactions serially). For example, a billing application cannot compute the total cost of a bill before it has looked up the rates that apply to the customer and computed subtotals for each different category of services. However, in SOA the sequence of transactions relies on the dependencies between services. For example by looking at figure 1 we can see that service S4 can't be executed before services S2 and S3 are executed. This means that the order in which STPs are executed must be as follows: STP_3, STP_4, STP_2, STP_1. The following rule captures this and guarantees the global consistency of all services and data items. **Certification Rule 8:**

For each TP, the order in which STPs are performed must be certified to maintain the global consistency of all the services and data items.

2.2.7 Service Dependencies

Service dependencies can take place between any number of services in order to fulfill a single request, and as a result increases the number of data items being manipulated raising the probability of putting the system in an invalid state due to failure in one or more of the sub-services. To make the process of ordering, auditing and recovery possible each service must maintain a dependencies table that records all dependencies between different services in a service network as shown in the following rule. It is also possible to have one service be responsible for maintaining this dependencies table.

Enforcement Rule 4:

Each service must maintain a dependencies table recording all dependencies between different services in a service network in the abstraction of: (Service ID, Depends on (Sa ID, Sb ID, Sc ID)).

2.2.8 Auditing

Many CDIs are involved in the fulfillment of a request. Validating these CDIs after each step is not a convenient process nor does it guarantee that the overall system is in a valid state. Therefore, if a sub-service failed to respond to a service call due to any reason, there should be a recovery mechanism to roll back all manipulations done before the failure in order to return the system to the previous valid state it was in.

Certification Rule 9:

All TPs must be certified to write to an append-only CDI (the log) all information necessary to permit the nature of the operation to be reconstructed.

2.2.9 Integrity Verification and System State

Whenever all the CDIs meet the integrity constraints of the system, the system is said to be in a valid state. The IVPs are responsible for checking that all CDIs in the SOA network are is in a valid state before the beginning of any new transaction.

Certification Rule 10:

All IVPs must properly ensure that all CDIs are in a valid state at the time the IVP is run. In the case of performing a business process in a SOA environment the IVP of the root service is valid if the IVPs of all sub-services are valid.

Certification Rule 11:

A well formed service (WFS) must be certified to ensure that all sub-service CDIs remain in a valid state and that the global consistency of CDIs is valid despite failure of any service or sub-service.

2.2.10 Filtering

The original CW model requires that all inputs whether from users or responses from other services be filtered at the interface before being used. The filtering process filters the data items into CDIs or UDIs based on how they affect the integrity of the system. All inputs entered by users to the services are considered UDIs and needed to be upgraded to CDIs or otherwise rejected.

Certification Rule 12:

Any TP or STP that takes a UDI as an input value must be certified to perform only valid transformations, or else no transformations, for any possible value of the UDI. The transformation should take the input from a UDI to a CDI, or the UDI is rejected.

If a TP started with a valid state and all certification and enforcement rules were applied, then it is guaranteed that none of the services will enter a bad state due to any reason.

2.3 Relating SCWIM to SOA Technologies

Applying SCWIM model to existing technologies is a challenging process. However, it is easier to apply a set of the rules instead of applying all of the rules at once to one existing SOA standard. This can be done with respect to the functionality of the SOA standard or technology being modified. For example SOA standards dedicated to authentication or authorization (e.g SAML,XACML) can make use of authentication and authorization rules. To facilitate this process, SCWIM's rules were grouped into nine different categories as was shown in the previous section. These were: encapsulation, service contract, concurrency and consistency, authentication and authorization, separation of duty, transaction sequencing, service dependencies, integrity verification and system state, auditing, and filtering.

Encapsulation and Service Contract certification rules, make basic demand of SOA implementations that are essentially definitional. The requirements imposed demand that services have contracts that explicitly describe data, integrity, and other issues. These are good practice and to some extent inherent to the SOA philosophy.

Concurrency, consistency, and transaction sequencing, are related to each other in that they all affect the global consistency if not well managed. The set of rules associated with them are used to manage the performance of transactions to guarantee global consistency of all services and CDIs.

Recent work in [26, 27, 28], provides some solutions to the transaction concurrency problem in web services environment. In both [27, 28] the proposed solutions guarantees the global correctness of concurrent transactions by allowing direct communication between coordinators of dependent transactions. Whereas in [26], direct communication between transaction coordinators is avoided by the use of a participants manager that maintains a conflict matrix which is used to detect any dependency between

concurrent transactions. Preventing data disclosure by keeping the information about business transactions restricted to the coordinators which are responsible for them.

Implementers of SOA technology must obey the concurrency and consistency rules regardless of technology. It seems implementers currently use ad hoc approaches such as manual scheduling transactions to solve concurrency problems. As application consistency constraints will vary, the certifier must verify that the chosen approach is sufficient to obey.

We believe that SCWIM concurrency, consistency, and transaction sequencing rules can work as a base for any future solution, and if carefully applied to any existing SOA technology can guarantee global consistency. We are unaware of any integrity verification or system state validation technology being used, but we believe that our model can guide the development of a standard or a technology in this regard.

Filtering of inputs to services can be based on user id, source address, nature of request, trust level, etc. In our model, filtering of inputs was based on how these inputs affect the overall integrity of all services and data item into CDIs and UDIs. Other forms of filtering can compare the inputs to black lists or white lists depending on the input type or source. In the case of integrity we believe that filtering the inputs into CDIs and UDIs is more convenient and serves the purpose of the model.

In SOA, different layers of filtering can take place: syntactic filtering (e.g XML parser, which ensures that no arbitrary inputs are entered to the service and that all inputs meet the message's format and structure) and semantic filtering (e.g detecting SQL injection).

A lot of work have been done on authentication and authorization of SOA (e.g SAML, XACML, ABAC [17, 15]). Authentication and authorization can be local to the service or centralized. We believe that our model's authentication and authorization rules can be used in any environment and for the development and modification of any SOA standard or technology. If these rules were paired with the separation of duty and filtering rules they will form a complete authentication and authorization system.

The audit information obtained by the mechanism certified in certification rule 9 can be used in different ways and by different technologies. One way to use it is in developing an analysis tool that will determine the degree of trust based on previously performed transactions. Another way would be to verify that all of the certification and enforcement rules have been successfully executed and applied to all data inputs.

CHAPTER 3. SOA Testbed

3.1 Introduction

Looking for a comprehensive SOA testbed was not an easy process. For a SOA testbed to be considered a useful testbed, it should fulfill the following requirements: be reusable, open source, extensible, come with large data sets, be cross platform. Until the moment of writing this thesis we were unaware of any SOA testbed that has all of these requirements other than the WorldTravel testbed. Many SOA related applications, implementations and tools were behind the development of this testbed such as RUBiS, Java Adventure Builder, Nutch, Intel Mash Maker, Yahoo Pipes, Apache Tuscany, httperf and StreamGen [29].

The first part of this chapter gives a brief description of the testbed architecture, setup, corrections and modifications. Whereas the second part discusses the components used in the development of our specification based intrusion detection system along with the monitoring tool used for capturing and monitoring the SOAP traffic between different services in the SOA network.

3.2 WorldTravel System Architecture

WorldTravel system [29] is an open source SOA testbed. This testbed resembles the travel industry system and it is a simplified version of WorldSpan, a GDS whose users include Delta air lines, Expedia, Orbitz, Hotwire, and Priceline. The GDS is short for global distribution systems which is responsible for providing services such as pricing and ticket sales for travel agents or customers. The testbed has the following components; the Travel Website (TWS) which is the interface necessary to help users look for fares, the load generator which represents the customer and sends requests to the travel web site, and the global distribution systems (GDS) which is the heart of the testbed. This component consists of three internal components, these are a database server (DB), a query node, and a load balancer. The

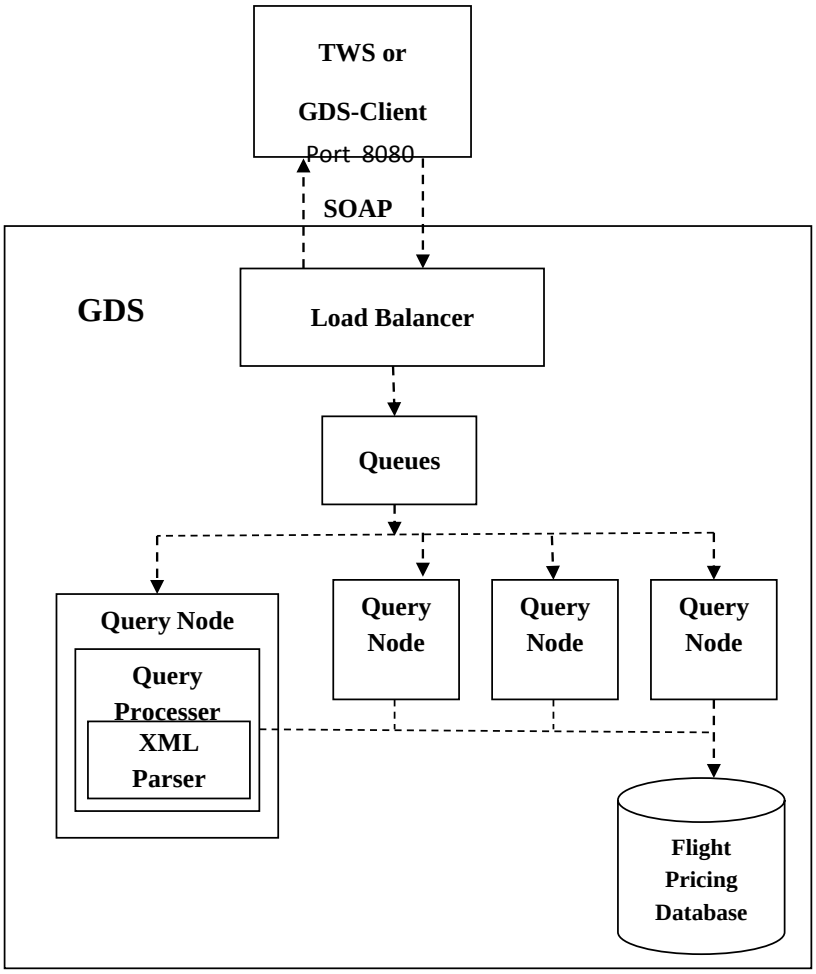

Figure 3.1 The original worldtravel SOA testbed architecture

architecture of the testbed is explained in details in [29]. The internal architecture of the GDS and how it interacts with other service is shown in figure 3.1. Each of the testbed components is independent and meaningful on its own and does not depend on other components or applications to use it. This characteristic shows the beauty of SOA and the endless possibilities that can take place.

Different open source software like apache Geronimo server and MySQL were used in the development of the testbed, as well as a set of communication and messaging standards such as SOAP messages and java messaging service (JMS) used for the communication between services. The WorldTravel SOA testbed came with a large database taken from WorldSpan Inc database more details can be found in [29].

The testbed services interact with each other when a customer searches the travel web service TWS

for an airline ticket or fare. As mentioned earlier, the testbed's main building blocks are the GDS and the travel website, these two services are the only parts of the testbed that use SOAP messages to communicate with each other. The GDS service contains the load balancer, one or more query nodes and the FlightPricing database. The load balancer is the front end of the GDS service and the part that communicates using SOAP messages, accepts requests, and returns responses once they have been processed, and communicates with the query nodes using queues, the query processing nodes are responsible for polling data from the database on demand to fulfill requests. The customer is represented using a load generator which generates requests to the travel web site. More details can be found in [29].

3.2.1 World Travel Testbed Setup

The WorldTravel testbed was setup using VMWare under Linux operating system. A minimum setup of five virtual machines was used, one for each of the following services: Travel Website (TWS), Global Distribution Systems (GDS), Query Processing (QPS), GDS client and finally the DB server. More details about how these parts interact exactly with each other can be found in WorldTravel original paper [29].

3.2.2 World Travel Testbed Corrections

The original WorldTravel testbed system went through some corrections to get it to work properly. Two main corrections were made to the original testbed. First, we couldn't get the testbed to work even though we followed all the steps given by the original developers of the testbed as shown in appendix A. After thorough investigation through all files and services we discovered that the GDS service was missing an ejb.jar file. This file is necessary and it contains the XML deployment descriptor. To solve this problem we wrote our own file. The content of ejb-jar.xml file is listed in code list 3.1. Second, we had to change the referenced database columns used in the code to match the column names available in the database.

3.2.3 WorldTravel Testbed Modifications and Additions

The testbed in its original form and components provides a raw platform for researchers and students to experiment with, change or extend. As we mentioned earlier we are developing a specification-

Listing 3.1 ejb-jar.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/ejb-jar_2_1.xsd" version="2.1">
  <display-name>Generated by XDoclet</display-name>
  <enterprise-beans>
  <message-driven>
    <ejb-name>
      FlightPricingQueryMDB
    </ejb-name>
    <ejb-class>
      edu.gatech.cercs.soa.gds.ejb.FlightPricingQueryMDB
    </ejb-class>
    <transaction-type>Container</transaction-type>
  </message-driven>
  <message-driven>
    <ejb-name>FlightPricingResultMDB</ejb-name>
    <ejb-class>
      edu.gatech.cercs.soa.gds.ejb.FlightPricingResultMDB
    </ejb-class>
    <transaction-type>Container</transaction-type>
  </message-driven>
  <message-driven>
    <ejb-name>FlightPricingResultStateMDB</ejb-name>
    <ejb-class>
      edu.gatech.cercs.soa.gds.ejb.FlightPricingResultStateMDB
    </ejb-class>
    <transaction-type>Container</transaction-type>
  </message-driven>
  </enterprise-beans>
  <assembly-descriptor >
  </assembly-descriptor>
</ejb-jar>
```

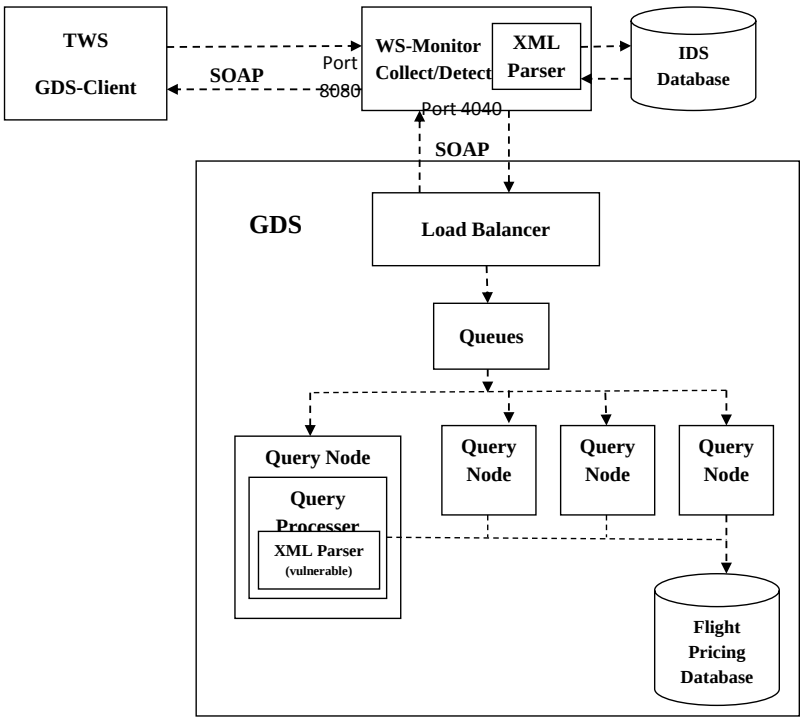

Figure 3.2 The modified WorldTravel SOA testbed architecture

based intrusion detection system for SOA networks, in order to do that we need to monitor the behavior of the services participating in it and develop a set of specifications that resemble these behaviors. At this point of our research we are only monitoring the behavior of the services that use SOAP messages for communication with other services. The following figure shows the modified WorldTravel SOA testbed architecture. The following subsections will discuss the modifications and additions applied to the testbed.

XML Parsers

We are emphasizing in this chapter that XML parsers are being used in this study because we are mainly studying XML injection attacks to study the effectiveness of our intrusion detection system. There are two main types of XML parsers in java. The Simple API for XML (SAX) parser and the Document Object Model (DOM) Parser.

For the purpose of simplicity and ease of use in our work we choose to use the DOM Parser. We are using it in two different places to achieve two different jobs as will be discussed later. Even though

DOM parser is not the fastest or the one with more memory efficiency, but it is easier to learn and it gives faster development results and in our case it was easier to create vulnerabilities using DOM parser. we need this parser to convert the document from a stream of data or bytes to a set of variables and values. At one point we had to write our own parser because the built in parser of the testbed was not vulnerable to attacks as discussed earlier in section 3.2.3.We needed to have an exploitable parser to be able to study the effect of different attacks on the behavior of the studied service.

As we mentioned earlier we are using two XML DOM parsers, the first is used for parsing the data intercepted by the wsmonitor tool during different stages of the IDS development, this parser is part of the GDS service. The second parser is in the QPS service, it parses and converts the request from an XML or SOAP document to a template that is used to create the SQL command for the original query. This second parser was changed from SAX to DOM.

Modifications and Additions List

Based on all of these issues and to fulfill the needs of our research we had to do the following modifications and additions:

1. Inserting a SOAP monitoring tool into the testbed to capture the behavior of the services as we will discuss in the next section. The tool we are using is called wsmonitor [30]. We developed several variations of it to cover different needs in different stages of the specification-based intrusion detection system development discussed in more details in chapter 4. These variations are: The Wsmonitor-Collect version and the Wsmonitor-Detect version. These variations will be discussed in more details in sections 4.4.1 and 4.4.3. The modified and newly added source code to wsmonitor in both cases is listed in appendix A and C.

2. Creating three databases that are saved in the GDS service machine. The first database will contain the data collected from Wsmonitor-Collect to be used later in the learning stage. The second database will contain the learned specifications. Finally the third database will have the result of the detection process. These databases will be discussed in more details later in section 3.4.

3. Writing our own XML parser for the GDS service machine since the parser that comes with the

Listing 3.2 Sample XML request

```
<FlightPricingQuery>
<Itinerary>
    <Trip>
        <From>JFK</From>
        <To>LAX</To>
        <Date>2011-02-27 10:00:00</Date>
        <NonStop>yes</NonStop>
    </Trip>
</Itinerary>
<Passengers>
    <Adult>2</Adult>
    <Child>1</Child>
    <Senior>3</Senior>
</Passengers>
<FareClasses>
    <FareClass>Economy</FareClass>
</FareClasses>
<Airlines>
    <Airline>AA</Airline>
    <Airline>DA</Airline>
</Airlines>
</FlightPricingQuery>
```

original testbed code is not vulnerable and we need to be able to apply some attacks on the services to test the effectiveness of our proposed approach.

All requests are sent to GDS service as XML documents. We will explain how this parser works by showing an example (see the XML document below). Assume that the GDS service received this simple request. Please note that actual requests are usually much larger than this example and contain more XML tags than what is shown in code list 3.2. This request asks for a flight from JFK airport to LAX airport on February 27th 2011 at 10:00AM for two adults and one child and 3 seniors on AA airline or DA airline.

The GDS service will have an empty template ready to be filled using the received XML document. The template will be filled as shown in table 3.

If an XML tag is injected in the XML document shown in code list 3.3. The template in the

From	JFK
To	LAX
Date and Time	2011-02-27 10:00:00
Non Stop	Yes
Adult Passengers	2
Child Passengers	1
Senior Passengers	3
Fare Class	Economy
Airlines	AA DA

Table 3.1 A filled template from the request sent to the GDS server.

GDS service for the attack code shown in code list 3.3 will be as shown in table 3. Note that ”From” field is now empty when ”Attack” is injected in ”From” field. The original XML parser on GDS service is not vulnerable to XML injection attacks. To make the GDS service attackable we rewrote the GDS XML parser code. After the modification we did, the filled template when an attack data is received will look like the filled template shown in table 3.

The behavior of the system when an attack message can be easily changed. Although the way

From	
To	LAX
Date and Time	2011-02-27 10:00:00
Non Stop	Yes
Adult Passengers	2
Child Passengers	1
Senior Passengers	3
Fare Class	Economy
Airlines	AA DA

Table 3.2 A filled template from the attack request sent to the GDS service. The service is not vulnerable in this case.

the system behaves is not that critical when such a simple attack is received, it just shows that, in principle, an XML injection attack is now possible regardless of how much harm/damage it may or may not cause.

Another thing we want to emphasize here is that we are using XML injection just as an example in this intrusion detection study. It is possible to make the system vulnerable for many other types of attacks. Studying the behavior of the service under such attacks is part of our future work plan.

Listing 3.3 Sample XML injection

```
<FlightPricingQuery>
<Itinerary>
    <Trip>
        <From>JFK<Attack>AttackData</Attack></From>
        <To>LAX</To>
        <Date>2011-02-27 10:00:00</Date>
        <NonStop>yes</NonStop>
    </Trip>
</Itinerary>
<Passengers>
    <Adult>2</Adult>
    <Child>1</Child>
    <Senior>3</Senior>
</Passengers>
<FareClasses>
    <FareClass>Economy</FareClass>
</FareClasses>
<Airlines>
    <Airline>AA</Airline>
    <Airline>DA</Airline>
</Airlines>
</FlightPricingQuery>
```

From	JFK<Attack>AttackData< /Attack>
To	LAX
Date and Time	2011-02-27 10:00:00
Non Stop	Yes
Adult Passengers	2
Child Passengers	1
Senior Passengers	3
Fare Class	Economy
Airlines	AA DA

Table 3.3 A filled template from the attack request sent to the GDS service. The service is vulnerable in this case.

3.3 Monitoring SOAP Traffic

Many tools are available for the purpose of monitoring SOAP traffic between a sender and a receiver such as the SOAPUI, membrane, XMLBus, wsmonitor [30] and many more. We need a tool that is platform independent, capable of handling multiple messages and most of all capable of intercepting SOAP messages. In the search for the perfect tool we found that the Wsmonitor [30] tool captures all the requirements and needs of our research. More details about this tool, its configuration and modification is given next.

3.3.1 Wsmonitor Tool

Wsmonitor (Web Services Monitor), is an open source, easy to use tool capable of capturing and monitoring SOAP messages and HTTP headers between a sender and a receiver. The tool uses port forwarding to capture the messages and displays them in a graphical user interface [30].

This tool is cross platform and multi-threaded so it can receive new requests while processing previously received ones. It was developed using java which means two things, it needs no memory management as apposed to using C++ and it is the same language used in the implementation of the WorldTravel testbed we are using, making it easier for us to incorporate it in the testbed. All monitoring, parsing, learning and detecting takes place on the GDS server. We want to learn the specifications and behaviors of this service since it is the only service in the testbed that uses SOAP messages.

3.3.1.1 Wsmonitor Configuration

The wsmonitor tool needs configuration when it comes to specifying the values for the listen port, target host and target port. These values determine what port the tool will listen on for incoming new messages to be intercepted and then forwards those intercepted messages to the specified target port in the target host. The wsmonitor tool has an XML-based configuration file where the listen port, target host and target port are specified. If this configuration file was not available, a default value of ”8080” for listen port, ”localhost” for target host, ”4040” for target port is assumed.

This tool is originally designed to Capture SOAP messages and HTTP headers and display them in a graphical user interface. We modified it to intercept the SOAP message and parse it before it reaches the GDS service and we made some modifications to the configuration file. The listen port is changed to 4040 and the target port to 8080. This means that the monitored service, GDS in this case, should run on port 4040 rather than the default 8080 in order for any traffic to go through port 4040 and wsmonitor first before being forwarded to the GDS service. Wsmonitor can run on the same machine as the GDS or on a different machine. All ports on the GDS server should be blocked using a firewall and allow only traffic going to port 4040 to force all communications to go through wsmonitor first.

3.3.1.2 Wsmonitor Modification

The wsmonitor tool software was modified to not only show the intercepted message, but also to be able to save it to a database as well, for later analysis. As mentioned earlier, several variations of the tool were needed throughout the development of our IDS to achieve two additional functions in addition to intercepting SOAP messages and forwarding them. The first function would be to collect data from the intercepted messages and then to save this data to a database for later analysis. The second function would be to detect any possible intrusions related to change in the behavior of the monitored service. For the later function to work properly, a set of specifications need to be extracted, learned, and developed using the collected data from the first function. Three programs were written to perform these functions as listed below, two are related to wsmonitor and a third responsible for learning the specifications of the service from the messages collected. It is a separate program that is not related to wsmonitor. These programs are:

- Wsmonitor-collect program: responsible for collecting data for the learning phase. See section 4.4.1 for more details and appendix A for the source code.
- Learning-phase-IDS program: responsible for learning specifications and behaviors. See section 4.4.2 for more details and appendix B for the source code.
- Wsmonitor-detect program: responsible for detecting potential intrusions. See section 4.4.3 for more details and appendix C for the source code.

Each program will be discussed in more details in the chapter 4 when we talk about each IDS development phase.

3.4 Testbed Databases

The original testbed has a huge database called the FlightPricingDB built in its architecture as we mentioned earlier. Our focus here in this section is on the databases we need to create to satisfy the storage needs for different stages in the development of our IDS. For that purpose three new databases were created:

- The first one is the LearningPhaseDataDB responsible for holding the data that will be used in the learning phase later. This database consists of two sets of tables for a total of four database tables. The first set is used for saving the raw data of both request and response messages and the second set is used for saving the parsed data for the same request and response messages. The first table is the SOAPIDSRequestTable used to save the raw request message data with the following columns:
  - Request ID: this is the primary key for this table and it is the time stamp of when this request was intercepted.
  - Requesting IP: this is the IP address of the machine sending this request.
  - Requesting Port: this is the port number from which the request was sent.
  - Requesting HTTP header: this is the http header of the message.
  - Requesting SOAP message: this is the SOAP message body.

- Requesting Time: this is the same as the request id.
- Request Attachment: it is a Boolean that indicates whether a request has an attachment or not.
- Request Length: it tells the length of the received SOAP message.
- Request Encoding: this variable tells the type of character set encoding used.

The second table is the response message table called SOAPIDSResponseTable which has the same columns in the request message table above, but with the exception of changing ”request” in the column names to ”response”.

The second set of tables used for saving parsed data has two tables one for the parsed request messages and the other is for the parsed response messages. The third table called the SOAPIDSRequestVarsTable which, as stated earlier, is used for saving the parsed intercepted request message data and it contains the following columns:

- Parse Time: this variable tells when the request was received to be parsed. it works as a primary key as well.
- Request ID: this is the primary key that will connect this table to the first table. It is the same as the parse time, but one millisecond is added for each tag parse time to keep it unique. There could be better things to use as a primary key, but for now this choice seems to be good enough as did not cause any problem through our study.
- Request Var Type: the type of the XML variable whether it is a #text or #comment ... etc.
- Request Var Name: the name of the request XML tag.
- Request Var Value: the value of the request XML tag.

The fourth table is the SOAPIDSResponseVarsTable which is dedicated for saving the parsed response messages data. This table is the same as the SOAPIDSRequestVarsTable but with the exception of changing the column name from request to response.

- The second database is the LearningDB which consists of 20 tables, so far, that summarize the learned specifications. These specifications must be learned for both the request and the response

messages and then saved. As mentioned earlier, the data used in this stage is taken from the first database LearningPhaseDataDB. The service specification will be extracted and learned through this stage as will be discussed in the next chapter. The LearningDB database stores the learned service specifications such as the following: the variables names, data length, encoding list, SOAP length, variables count range, if data is Boolean, or number or date or if it has special characters, and finally learn the relationships between all requests and responses by learning what request initiated each response. Here is a list of the request message specifications tables:

  - ReqTagsNames.
  - ReqDataLength.
  - ReqDataIsBool.
  - ReqDataIsDate.
  - ReqDataIsNum.
  - ReqDataHasChar.
  - ReqSOAPLen.
  - ReqEncodingList.
  - ReqNameCountRange.

  We have the same set of tables for the response message specifications. The remaining two tables are the most important tables in the learning phase. These tables are the CallsSequenceAND and the CallsSequenceOR. The first table lists which responses are always proceeded by which requests. The second table lists the relations between each response and which requests may have initiated it.

- The third database is used in the detection stage and it is called the DetectionPhaseDB. The tables in this database are the same as the tables in the LearningPhaseDataDB, since we need to learn the specifications of the new intercepted data and then compare it later with the previously learned specifications stored in the LearningDB.

More details about all of the functions necessary to fill up these tables will be given in chapter 4 and in the appendices attached to this thesis.

CHAPTER 4. Specification Based Intrusion Detection System for SOA Networks

4.1 Introduction

We live in a world of services that are widely used both by humans and applications. Making sure that these services are secure, and that all transactions or messages coming in or out of these services are also secure is a challenge. In this chapter, we are proposing a specification-based intrusion detection system (IDS) capable of detecting intrusions based on abnormal behaviors of the monitored service. In this chapter, we summarize some of the related work in this area, then we discuss the process of developing our specification based IDS.

4.2 SOA Intrusion Detection Systems

Many Intrusion detection systems have been developed for the purpose of detecting unauthorized or misused privileges or actions in a system, whether this system consists of one computer or many on the same network or on different networks. The detection mechanisms fall into one of the following categories:

- Anomaly based intrusion detection: looks for behavior that deviated from normal system use. It can identify previously unknown attacks, but it has a large number of false positives.
- Misuse based intrusion detection: looks for behavior that matches a known attack scenario. It is efficient with few false positives, but it detects only previously known attacks.
- Specification based intrusion detection: in this detection mechanism specifications are used to characterize legitimate program behavior, and any deviation from these specifications is considered an intrusion. It produces low rate of false positives and it captures the strengths of both

misuse and anomaly detection mechanisms, but if the specifications were not developed accurately it can affect the accuracy of the IDS.

The stability and efficiency of an IDS depends on the observable used to distinguish between acceptable and unacceptable behaviors. Selecting a set of dynamic behavioral characteristics to monitor a service is a key design decision for an IDS. It will influence the types of analysis that can be performed and the amount of data that will be collected [31]. Several methods have been proposed for this purpose:

1. Methods that characterize the behavior of privileged processes or programs using:
   - Short sequences of system calls.
   - Program specifications or policies which require knowledge of the internals and intended role of a program.
   - System call arguments.
2. Methods that analyze network traffic.
3. Methods that characterizes the behavior of users by looking at user profiles generated by audit logs.

Monitoring the behavior of programs or services is more effective and more efficient because the behavior of services is limited and relatively stable compared to the range of behaviors users can have. Users perform a wider variety of actions, and these actions may change considerably over time and are usually unpredictable, while the actions or functions of services do not vary much over time. In the following discussion we will focus on the related work done in the area of monitoring program/service behavior.

To our knowledge, no existing IDS was developed with SOA networks in mind except for FIX (filter to inspect XML) model [12] which is an XML IDS. This model assumes that different XML filters are needed in different scenarios for the security inspection of XML-based applications. These filters inspect XML data traffic looking for XML structural anomalies and can be applied on a case by case basis depending on the payload anticipated by the application.

Early research work [31, 32, 33, 34, 35] focused on building privileged programs profiles by capturing short sequences of system calls. All of the IDSs proposed in these papers are anomaly based

detection systems. These systems usually rely on system call sequences to characterize the normal behavior of programs. Recently, it has been shown that these systems can be evaded by launching attacks that execute legitimate system calls sequences. The evasion is possible because existing techniques do not take into account all available features of system calls like system call arguments for example [36].

Another approach [36] analyzes program/service behavior by monitoring system call arguments without taking system call sequences into account. This IDS applies multiple detection models to system call arguments allowing the arguments of each system call invocation to be evaluated from several different perspectives. A model is a set of procedures used to evaluate a certain feature of an argument, such as the length of a string, structural inference, string character distribution, and token finder. Combining the anomaly score from these models into an overall aggregate score will determine whether an event is part of an attack or not. This method uses the Bayesian networks for the classification process instead of threshold which gives less false positives and more true positives. If an attack is carried out without performing system call invocations, without affecting the value of the arguments or using system call arguments that do not differ substantially from the values used during the normal execution then this approach will not be able to detect it.

Another available intrusion detection system for services [37] extends the application IDS model from considering only packet header information at the network and transport layer to include the application payload as well. Processing the payload of packets is not effective unless some knowledge of the application that creates them is available.

To distinguish the intrusive behavior, different classification measures were used in the previously discussed models, such as:

- The hamming distance.
- Cross-correlation.
- Hidden Markov model (HMM).
- Neural networks.
- Frequency based methods.
- Enumerating sequences.

- Finite state machine.
- K-nearest neighbor.
- Data mining approaches.
- Bayesian networks.
- Decision trees.

The work described in [38] proves that specification based IDS combine the strengths of misuse detection (accurate detection of known attacks) and anomaly detection (ability to detect novel attacks) and shows that specification based techniques can detect known as well as unknown attacks while maintaining a very low rate of false positives. In the coming sections we will discuss our proposed specification-based IDS for SOA networks.

4.3 Contributions

As mentioned earlier in this thesis we propose a specification-based IDS for SOA networks capable of detecting intrusions that affect the behavior of services. We assume that all the services studied in our system are web based services that use SOAP messages to communicate with each other. The detection technique used is based on the assumption that any change in the behavior of a service is an intrusion if this behavior does not meet a set of known specifications developed for this service.

Our novel approach will provide the following advantages over the existing IDS for services:

- Our proposed IDS does not require knowledge of the underlying service code. Which makes it easy to implement.
- Our proposed IDS can detect all abnormal behaviors of the monitored service that might lead to an attack.
- Our proposed model can adapt to any changes in the service implementation, and will still be effective regardless of the programing language or platform used, which makes it flexible and implementation independent.
- We believe that our IDS will give a low false negative/positive rate.

4.4 Specification-Based IDS Development

Now that the SOA testbed of choice WorldTravel system is up and running and well configured, see chapter 3, we are ready to start talking about the stages necessary in the development of our specification-based intrusion detection system.
A service that uses our model of intrusion detection has to go through the following stages:

- Attack-free data collection phase: During this phase the IDS will collect a data that is supposedly clean from attacks. We advise that this data set be as large as possible to better profile the service.
- Specifications development and learning phase: during this phase the IDS will try to profile the data and learn its characteristics. The accuracy of the learned characteristics will depend mainly on the size of the data set used.
- Actual deployment and threats detection phase: Once the IDS has learned the service characteristics, it will now compare every captured message with the learned characteristics. Any deviation from the learned characteristics indicates a possible attack.

We will now discuss the phases mentioned above in more details.

4.4.1 Data collection stage

Two different data sets should be collected throughout the development of our IDS:

- Data used for the development of service specifications (learning phase).
- Data to be tested for intrusions (testing or deployment phase).

The first data set is the learning phase data. Once this set of data is collected, it is used to profile the service and develop a set of specifications for it. These specifications will then be used in the testing phase to test the legitimacy of actual captured behaviors.

Data for both of these sets can be drawn from different sources such as: web transactions records, SOAP messages or a dynamic link library. It is required that the data used for specification development in the learning phase be taken from a controlled environment free of intrusions to maximize the intrusion detection rate. Examples of this data include listing the functions called from the dll library by a specific

service. Knowing the order in which these functions were called can help in developing a specification for this service behavior.

It is necessary that data for both the learning phase and the testing phase be taken from the same data source. For example, if the specifications were developed based on data taken from SOAP messages and http headers, then data to be tested must be taken from SOAP messages and http headers. There is no need to understand the underlying service code to be able to develop specifications for services since these specifications do not depend on code details but rather on behavior related details.

4.4.1.1 Implementation

To test our specification based intrusion detection idea we chose WorldTravel testbed, see chapter 3. We decided to monitor the behavior of the service by monitoring the characteristics of the SOAP messages and the http headers communicated between the various parts of the testbed.

As a starting point in implementing the first phase of the intrusion detection process, namely the learning phase, we started with wsmonitor as a nucleus for our program. Wsmonitor is an open source java-based tool that intercepts SOAP messages and http headers communicated between two points. See chapter 3 for more details. However, in order to fit our needs more precisely, we did the following changes to wsmonitor. We called the new modified tool wsmonitor-collect, the source code of wsmonitor-collect is listed in appendix A.:

- The program was modified to log http headers and SOAP messages into separate files in a specific folder in the file system. The source code is listed in appendix A.2.
- The same captured traffic is also saved to a database, we called it LearningPhaseDataDB, for more convenient access later during the learning process. See appendix A.1 and A.4 for more details about this process.
- Wsmonitor-collect was setup such that it receives any traffic intended for the monitored service, GDS service in this case, then it processes the collected traffic and forwards it to its original destination. The process is a multi-threaded process where the forwarding process is done on a separate thread from the XML parsing process and characteristics collection. This enables a better and more efficient real time detection. See appendix A.2 for detailed code.

- The captured messages are then parsed into XML tags and their values and some packet characteristics are extracted from the http header. In particular, we collected the following data for each captured message (see appendix A.5 and A.6):
  - The system time at the moment of capturing the message. We used this value as a message ID.
  - The client/request source IP address and port number.
  - Message encoding type from the http header (see appendix A.2).
  - Message length. The length is not taken from the http header. The length used is the one we got from actually measuring the length of the string that represents the message itself.
  - SOAP/XML messages exchanged with the service. Each captured message is then parsed to get the name, value, and type of each XML tag. The result of the parsing process is also saved in an SQL database in a table of three columns (name, value, type) for easier processing later. The source code used to create this database is listed in appendix A.1.

All of the data collected in this stage is sane data collected from the testbed while it was up and running in a controlled environment free of vulnerabilities and attacks. The database created to hold this data has four tables, two for the request data and the other two are for the response data. More details about these tables and the database were given in chapter 3. The source code of the first phase of the intrusion detecting process, wsmonitor-collect or data collection phase, is listed in appendix A.

4.4.1.2 Example

In this example we will discuss how the SOAP requests are parsed. Parsing the http headers and logging the source port and IP address are relatively easy tasks to do and consequently we will not discuss them in detail here. The java source code we used for logging http header data is listed in appendix A.

A sample legitimate SOAP request sent to the GDS server is shown in code list 4.1 and a sample legitimate response to this request is shown in code list 4.2. When this request is received by wsmonitor-collect, it will be saved in an XML file and the whole message will be saved in the request SOAP messages SQL database. The message will then be parsed. The parsing result will be saved in the

parsed requests SQL database. The result saved in the database will look like the data shown in table 4.1. The parsed response result for the response in code list 4.2 is shown in table 4.2.

This process is done for every request/response that goes through wsmonitor-collect. For a better intrusion detection result this sane data set should be as large as possible and representative of the actual real world data. It should be as various as possible. This will help in a better characteristics learning process as will be discussed next.

Listing 4.1 Sample legitimate XML request

```
<FlightPricingQuery>
    <Header>
        <CustomerId>www.iastate.edu</CustomerId>
        <QueryId>2011-01-14 18:55:50 230</QueryId>
        <QueryMode>poll</QueryMode>
        <SearchId>3</SearchId>
        <SearchTimeStamp>398375989234587</SearchTimeStamp>
        <Expiration>15000</Expiration>
    </Header>
<Itinerary>
    <Trip>
        <From>JFK</From>
        <To>LAX</To>
        <Date>2011-02-27 10:00:00</Date>
        <NonStop>yes</NonStop>
    </Trip>
    <Trip>
        <From>LAX</From>
        <To>JFK</To>
        <Date>2011-03-27 10:00:00</Date>
        <NonStop>yes</NonStop>
    </Trip>
</Itinerary>
<Passengers>
    <Adult>2</Adult>
    <Child>1</Child>
    <Senior>3</Senior>
</Passengers>
<FareClasses>
    <FareClass>Economy</FareClass>
    <FareClass>Business</FareClass>
    <FareClass>First</FareClass>
</FareClasses>
<Airlines>
    <Airline>AA</Airline>
```

```
    <Airline>BA</Airline>
    <Airline>DA</Airline>
</Airlines>
</FlightPricingQuery>
```

Tag Name	Tag Value
FlightPricingQuery.Header.CustomerId	www.iastate.edu
FlightPricingQuery.Header.QueryId	2011-01-14 18:55:50 230
FlightPricingQuery.Header.QueryMode	poll
FlightPricingQuery.Header.SearchId	3
FlightPricingQuery.Header.SearchTimeStamp	398375989234587
FlightPricingQuery.Header.Expiration	15000
FlightPricingQuery.Itinerary.Trip.From	JFK
FlightPricingQuery.Itinerary.Trip.To	LAX
FlightPricingQuery.Itinerary.Trip.Date	2011-02-27 10:00:00
FlightPricingQuery.Itinerary.Trip.NonStop	yes
FlightPricingQuery.Itinerary.Trip.From	LAX
FlightPricingQuery.Itinerary.Trip.To	JFK
FlightPricingQuery.Itinerary.Trip.Date	2011-03-27 10:00:00
FlightPricingQuery.Itinerary.Trip.NonStop	yes
FlightPricingQuery.Passengers.Adult	2
FlightPricingQuery.Passengers.Child	1
FlightPricingQuery.Passengers.Senior	3
FlightPricingQuery.FareClasses.FareClass	Economy
FlightPricingQuery.FareClasses.FareClass	Business
FlightPricingQuery.FareClasses.FareClass	First
FlightPricingQuery.Airlines.Airline	AA
FlightPricingQuery.Airlines.Airline	BA
FlightPricingQuery.Airlines.Airline	DA

Table 4.1 Parsed request as saved in the SQL parsed requests database.

Listing 4.2 Sample legitimate XML response

```
<FlightPricingResult>
  <Header>
      <CustomerId>>www.iastate.edu</CustomerId>
      <QueryId>2011-01-14 18:55:50 230</QueryId>
      <QueryMode>async</QueryMode>
      <SearchId>3</SearchId>
      <SearchTimeStamp>398375989234587</SearchTimeStamp>
      <Expiration>15000</Expiration>
      <Status>complete</Status>
      <StatusDetail>Found a matching itinerary!</StatusDetail>
```

```
</Header>
<Itineraries>
   <Itinerary>
      <Price>
         <Fare>250</Fare>
         <Tax>30</Tax>
         <Fee>12</Fee>
      </Price>
      <Trip>
         <Stop>
            <From>JFK</From>
            <To>LAX</To>
            <Departure>2011-02-27 11:00:00</Departure>
            <Arrival>2011-02-27 16:00:00</Arrival>
            <Airline>DA</Airline>
            <FlightNumber>321</FlightNumber>
            <FareClass>Economy</FareClass>
         </Stop>
         <Stop>
            <From>LAX</From>
            <To>JFK</To>
            <Departure>2011-03-27 11:00:00</Departure>
            <Arrival>2011-03-27 16:00:00</Arrival>
            <Airline>DA</Airline>
            <FlightNumber>331</FlightNumber>
            <FareClass>Economy</FareClass>
         </Stop>
      </Trip>
      <Trip>
         <Stop>
            <From>JFK</From>
            <To>LAX</To>
            <Departure>2011-02-27 15:00:00</Departure>
            <Arrival>2011-02-27 19:00:00</Arrival>
            <Airline>DA</Airline>
            <FlightNumber>341</FlightNumber>
            <FareClass>Economy</FareClass>
         </Stop>
         <Stop>
            <From>LAX</From>
            <To>JFK</To>
            <Departure>2011-03-27 15:00:00</Departure>
            <Arrival>2011-03-27 19:00:00</Arrival>
            <Airline>DA</Airline>
            <FlightNumber>351</FlightNumber>
            <FareClass>Economy</FareClass>
         </Stop>
```

```
        </Trip>
      </Itinerary>
    </Itineraries>
</FlightPricingResult>
```

4.4.2 Specifications development stage

This stage is the most important and the most critical stage in the development of our IDS. Specification development means coming up with a set of rules that describe the expected behavior of different services in a SOA network or testbed. This stage depends on the learning data collected in the data collection stage where it is used to develop a set of specifications that characterize the behavior of all services in a SOA network. These characteristics can be learned by monitoring the behavior of different services and the transactions associated with them. For the purpose of our research we are monitoring the behavior of the global distribution systems (GDS) service only, since it is the only service that uses SOAP messages to communicate. Different characteristics can be learned about each service especially when it is in the process of completing a single transaction. Such characteristics will include, for example, the sequence of behaviors needed to fulfill a request, the frequency of occurrence of these behaviors and the type of behaviors and actions allowed...etc. The specifications developed for each service must describe the exact way in which this service will operate to fulfill a designated transaction. It is expected that collecting more learning data will lead to more convergence toward the ideal behavior and as a consequence less false alarms.

4.4.2.1 Implementation

This stage is implemented using the LearningPhaseIDS program (see appendix B). We need to learn the characteristics of the collected data from the previous stage, and extract all of the information that we can get out of the intercepted SOAP messages like the name of the variables or XML tags, the types of these variables, the number of occurrences of these variables (see appendix B.3), the minimum and maximum value of each variable if its value is supposed to be a number (see appendix B.8) or Boolean (see appendix B.10) or time (see appendix B.9), and the minimum length and the maximum length of each request/response message (see appendix B.6). To do that we need to run a set of tests against these variables to infer the type to see if it is one of the following: Boolean, date-time, number, and

finally check to see if there are any special characters in the XML tag (see appendix B.7). The special characters set that should be checked is specified earlier in the learning phase code.

We learned the types of each XML tag value by trying to convert its value to: Boolean, numerical, Date-Time. The same process is repeated for all values of every XML tag. If the conversion process to a specific type is successful for all values of a given tag, then that tag is of that type.

Later during the detection phase, any captured XML tag, for example, that is supposed to be Boolean when it is actually not, will be marked as a possible threat/problem. Any SOAP message whose length exceeds the maximum learned length or shorter than the shortest possible learned length will be marked as a possible problem/intrusion. Each suspected intrusion will be given a number that represents a threat level. The given threat level severity is usually based on experience and educated guesses.

Another characteristic that we checked is the encoding of each exchanged request/response message (see appendix B.5). A list of all encoding possibilities is then built. Any legitimate message later is expected to have one of the encoding possibilities found during the learning phase.

We believe that monitoring the frequency of XML tags in a SOAP message is of utmost importance (see appendix B.3). During the learning process we try to learn the minimum and maximum count of each XML tag in all SOAP messages. If it happens during the detection phase that a certain XML tag occurred more/less than it should then that might be an indication of a possible XML injection attack, for example.

We also monitored the length of every XML tag in every SOAP message. A minimum and maximum value of the length of every XML tag value is learned and saved in a database. The length of a legitimate XML tag value is expected to be within the learned range (see appendix B.11).

The next step in the learning process would be to learn the sequences of these variables for each request/response message pair to study the relationship between the request variables that initiated the response variables (see appendix B.4). This relationship can be a one-to-one relationship meaning that one response variable is caused by one request variable, or a one-to-many relationship where one response variable is caused by many request variables. Figure 4.1 represents these relationships.

For example, certain responses never appear unless a specific request is received. Getting a certain response from the service when the minimum requirement for how the shape of the request is, is an

indication of a possible attack. On the other hand, some responses never appear when a certain request is initiated. For example, getting a username or password when the request was about a flight data is a strong indication of an attack. In our implementation we call this behavior CallSequence. More details about this implementation can be found in appendix B.4.

A more abstract way to describe what we called CallsSequence is as follows: Assume that the set of all possible request and response XML tags is:

$$Req = \{V_1, V_2, ...V_m\} \tag{4.1}$$

$$Res = \{V'_1, V'_2, ...V'_n\} \tag{4.2}$$

where V_i and V'_j are request and response XML tags names correspondingly.

Assume that a request with request ID req_id_1 is a vector that can be represented as follows:

$$Request_{req\_id\_1} = \{V_1, V_2, ...V_p\} \; where \; V_i \in Req \tag{4.3}$$

where V_i is an XML tag name such as FlightPricingQuery.Itinerary.Trip.From or any XML tag name detected during the data collection phase. The first column in table 4.1 is a list of such possible values. This request $Request_{req\_id\_1}$ will result in a response $Response_{req\_id\_1}$ where:

$$Response_{req\_id\_1} = \{V'_1, V'_2, ...V'_p\} \; where \; V'_j \in Res \tag{4.4}$$

The same applies to the rest of the requests and responses:

$$\begin{aligned}
Request_{req_id_1} &= \{V_1, V_2, ...V_p\} \; where \; V_i \in Req \\
Response_{req_id_1} &= \{V'_1, V'_2, ...V'_q\} \; where \; V'_i \in Res \\
Request_{req_id_2} &= \{V_1, V_2, ...V_r\} \; where \; V_i \in Req \\
Response_{req_id_2} &= \{V'_1, V'_2, ...V'_s\} \; where \; V'_i \in Res \\
... &\quad ... \\
... &\quad ... \\
Request_{req_id_n} &= \{V_1, V_2, ...V_t\} \; where \; V_i \in Req \\
Response_{req_id_n} &= \{V'_1, V'_2, ...V'_u\} \; where \; V'_i \in Res
\end{aligned}$$

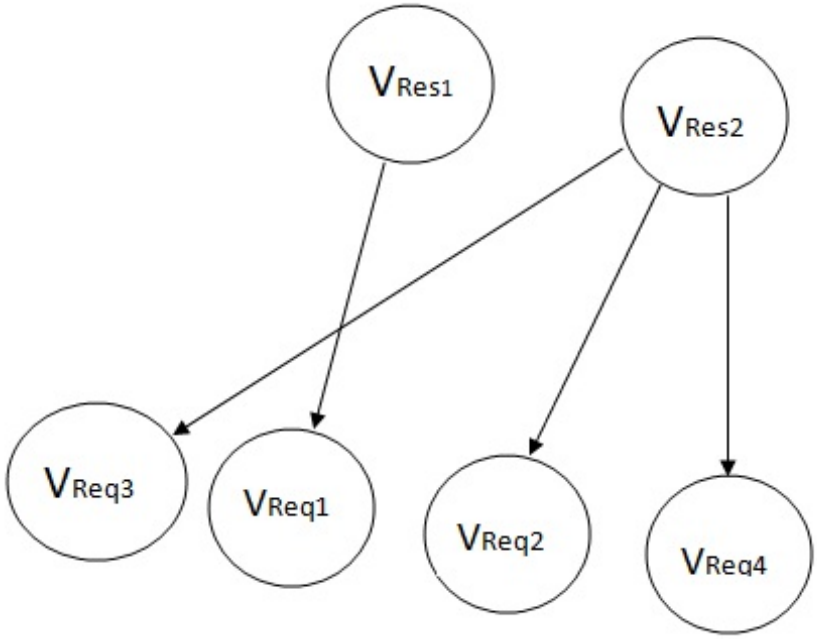

Figure 4.1 The relationship between request variables and response variables

We need next to isolate the requests that resulted in a response XML tag V_i'

$$\forall\, V_i' \in Res\ \exists\, RequiredSet_i = Request_{req\_id\_1} \cap Request_{req\_id\_2} \cap \ldots Request_{req\_id\_n} \tag{4.5}$$

where

$$V_i' \in Response_{req\_id\_1}.Response_{req\_id\_2}...Response_{req\_id\_n}$$

The same process needs to be repeated for all V_i' response XML tags. We should now have a $RequiredSet_i$ for every V_i'. Getting a response V_i' without having all members of the set $RequiredSet_i$ in the request XML message is a clear sign of unusual/intrusive behavior.

Another way of detecting possible intrusive behavior is to tabulate the list of requests that may precede a given response.

$$\forall\, V_i' \in Res\ \exists\, OptionalSet_i = Request_{req\_id\_1} \cup Request_{req\_id\_2} \cup \ldots Request_{req\_id\_n} \tag{4.6}$$

where

$$V_i' \in Response_{req\_id\_1}.Response_{req\_id\_2}...Response_{req\_id\_n}$$

Note that

$$RequiredSet_i \subset OptionalSet_i$$

and that for both sets

$$RequiredSet_i.OptionalSet_i \subset Req$$

Note that using $OptionalSet_i$ we can calculate the set of XML tags that cannot precede a given response. $ForbiddenSet_i$ is the compliment of $OptionalSet_i$. That is:

$$ForbiddenSet_i = OptionalSet_i^C \tag{4.7}$$

All of the relations between the request and response message pairs and the characteristics learned are saved in the database to be used later on to distinguish legitimate behaviors/relations from illegitimate ones. More details about the implementation of CallsSequence can be found in appendix B.4.

We would like to note here that our implementation of the learning process is not iterative, meaning that every time the learning program is run, the characteristics are extracted and learned while ignoring any previously learned data. It is worth knowing that the process of specifications learning and development must be well trusted and certified to give a comprehensive behavior characterization of the studied service.

4.4.2.2 Example

Once a large set of data is collected using wsmonitor-collect, the data characteristics should be learnt using the learning phase routine. For example it should be known after the learning process that FlightPricingQuery.Passengers.Adult is always a number. Its value does not exceed, depending on the collected data set, say 100. It does not contain any special characters. Whereas FlightPricingQuery.Itinerary.Trip.From is a string and its length does not exceed 3 characters. It also does not contain any numerical characters or special characters and we cannot infer a date from its value ...etc. The same learning process is run on every single XML tag. The length of SOAP request and response messages can also be learned. Any message cannot be less than the length of the smallest message and cannot exceed the length of the largest one... etc.

Lets take this simple example:

Listing 4.3 Simple XML injection attack

```
<FlightPricingQuery>
<Itinerary>
    <Trip>
        <From>JFK<Attack>AttackData</Attack></From>
        <To>LAX</To>
        <Date>2011-02-27 10:00:00</Date>
        <NonStop>yes</NonStop>
    </Trip>
</Itinerary>
<Passengers>
    <Adult>2</Adult>
    <Child>1</Child>
    <Senior>3</Senior>
</Passengers>
<FareClasses>
    <FareClass>Economy</FareClass>
</FareClasses>
<Airlines>
    <Airline>AA</Airline>
    <Airline>DA</Airline>
</Airlines>
</FlightPricingQuery>
```

There are several possibilities of catching this attack. One possibility is that the request SOAP message length may exceed that maximum learned length. Another possibility is that ”Attack” XML tag may not be in the set of possible XML tag requests learned during the learning phase because it did not exist during the sane data collection phase. Another possibility is that it may initiate a response that does not meet the CallsSequence criteria we described above. Another possibility is that the value of ”From” field in which ”Attack” is injected cannot contain any special characters, $<$ or $>$ in this case. Also the value of ”From” field length now exceeded its maximum possible length. More details and example of exchanged request and response messages from the testbed WorldTravel can be found in section 3.2.3.

4.4.3 Detection and classification stage:

Once these specifications are learned and collected for the target service in the testbed, the next step would be to classify the actual services behavior into either legitimate or illegitimate behaviors. To do so, records of actual behavior or testing data will be compared with the developed specifications. If they

do not match, then an intrusion will be assumed. The results of all comparisons will be displayed in the IDS interface in real-time, and then logged for further possible analysis and investigation.

4.4.3.1 Implementation

Classification of request/response legitimacy is done using the wsmonitor-detect program. This program is responsible for executing the comparison process between the specifications learned and saved in the database, and the data that need to be tested. When wsmonitor-detect is started it loads, for one time, the characteristics/specifications learned in the previous learning phase. Each intercepted SOAP message goes through the learning program to learn its specifications as discussed earlier, then these specifications are compared against the known specifications saved.

The result of the classification process can lead to know or unknown request and/or response. When either/both of the request or/and the response do not meet the learned characteristics then a possible attack/threat is detected. The result of the comparison process is one of these four combinations. Each one of these combinations represents a case and each case needs a different action to be taken by the IDS. The four possibilities are:

- Known request behavior with known response behavior: This is the case for normal and legitimate requests and responses, all XML tags and their values meet the characteristics learned during this stage. In this case both requests and responses are known to the intrusion detection system. The exchanged communication in this cases is assumed to be safe or attack free. We would like that all intercepted messages be of known requests and responses behaviors where no action need to take place. This means that we have seen this behavior before and that it is a normal behavior.

- Unknown request behavior with known response behavior: The actions to be taken against this case depends on different factors, one of these factors is to do a risk analysis, the risks behind each case needs to be evaluated and rated, the more serious the risk is the more action needs to take place.

- Known request behavior with unknown response behavior: This is a similar case to the previous case from a risk analysis point of view. However, it can be more severe because an unknown response from a service with well known behavior is more risky than an unknown input from

the user. The service behavior is more limited than the user's behavior which can be affected by human factors as such unintended/mistaken inputs.

- Unknown request behavior with unknown response behavior: This is the most severe case. Still, a detailed risk analysis needs to be done. Currently we are using a severity level that is based on educated guesses and observations. We tried to give a severity level that ranges from 1 to 4 for every unusual behavior. Unusual responses are given more severity level than unusual requests. Flags that are raised because of a special character inserted where it should not be is given more severity level than a flag that is raised because a tag value was alphabetic when it was supposed to be numeric. CallsSequence flags,see page 45, are given the highest severity level. Developing a more convincing and accurate risk analysis is part of our future work plan.

The source code for the part of wsmonitor-detect that is responsible for detecting unwanted requests is listed in appendix C.1. The corresponding part of the code that detects unusual responses is listed in appendix C.2. As stated above in section 4.4.2, one of the characteristics that we used is the request/response calls dependencies. The source code that takes care of this task is listed in appendix C.3.

4.4.3.2 Example

The discission presented in section 4.4.2.2 applies to this stage as well. We believe that the discussion presented there along with the discussion in section 3.2.3 are enough to serve the purpose of explaining the detection process.

4.4.4 Evaluation stage

The final step in the development of our IDS is to evaluate its effectiveness and performance. This can be done by using either the detection rate which is the ratio between intrusions detected and intrusions attempted, or by using the false alarm rate. These two rates can be represented together using a receiver operating characteristic (ROC) curve. We believe that if the data collected for the specifications development stage was taken from an intrusion free environment and that the specifications developed describes the exact behavior of all services in a SOA network, then we will get a high intrusion detection

rate and a low false positive/negative rate. A detailed study that demonstrates such results is part of our planned future work.

This stage has two main drawbacks in it. The first is that our specification-based-IDS was not evaluated because we didn't have a large learning/testing data sets to begin with. The second drawback is the fact that we tested our IDS on one type of attacks only which is the XML injection attack, this does not mean that other types of attacks are not possible for testing, its just that we didn't test against them. Testing against more types of attacks is also part of our planned future work.

Even though our proposed IDS can detect all attacks affecting the behavior of services and will give a low false positive/negative rate if accurate specifications were developed, it can not detect attacks that mimic/do not affect the service behavior such as denial of service attacks which can affect the availability of a service in a SOA network. Despite that, we still believe that our specification-based IDS will open the door for more research in this area in the years to come.

Tag Name	Tag Value
FlightPricingResult.Header.CustomerId	www.iastate.edu
FlightPricingResult.Header.QueryId	2011-01-14 18:55:50 230
FlightPricingResult.Header.QueryMode	async
FlightPricingResult.Header.SearchId	3
FlightPricingResult.Header.SearchTimeStamp	398375989234587
FlightPricingResult.Header.Expiration	15000
FlightPricingResult.Header.Status	complete
FlightPricingResult.Header.StatusDetail	Found a matching itinerary!
Itineraries.Itinerary.Price.Fare	250
Itineraries.Itinerary.Price.Tax	30
Itineraries.Itinerary.Price.Fee	12
Itineraries.Itinerary.Trip.Stop.From	JFK
Itineraries.Itinerary.Trip.Stop.To	LAX
Itineraries.Itinerary.Trip.Stop.Departure	2011-02-27 11:00:00
Itineraries.Itinerary.Trip.Stop.Arrival	2011-02-27 16:00:00
Itineraries.Itinerary.Trip.Stop.Airline	DA
Itineraries.Itinerary.Trip.Stop.FlightNumber	321
Itineraries.Itinerary.Trip.Stop.FareClass	Economy
Itineraries.Itinerary.Trip.Stop.From	LAX
Itineraries.Itinerary.Trip.Stop.To	JFK
Itineraries.Itinerary.Trip.Stop.Departure	2011-03-27 11:00:00
Itineraries.Itinerary.Trip.Stop.Arrival	2011-03-27 16:00:00
Itineraries.Itinerary.Trip.Stop.Airline	DA
Itineraries.Itinerary.Trip.Stop.FlightNumber	331
Itineraries.Itinerary.Trip.Stop.FareClass	Economy
Itineraries.Itinerary.Trip.Stop.From	JFK
Itineraries.Itinerary.Trip.Stop.To	LAX
Itineraries.Itinerary.Trip.Stop.Departure	2011-02-27 15:00:00
Itineraries.Itinerary.Trip.Stop.Arrival	2011-02-27 19:00:00
Itineraries.Itinerary.Trip.Stop.Airline	DA
Itineraries.Itinerary.Trip.Stop.FlightNumber	341
Itineraries.Itinerary.Trip.Stop.FareClass	Economy
Itineraries.Itinerary.Trip.Stop.From	LAX
Itineraries.Itinerary.Trip.Stop.To	JFK
Itineraries.Itinerary.Trip.Stop.Departure	2011-03-27 15:00:00
Itineraries.Itinerary.Trip.Stop.Arrival	2011-03-27 19:00:00
Itineraries.Itinerary.Trip.Stop.Airline	DA
Itineraries.Itinerary.Trip.Stop.FlightNumber	351
Itineraries.Itinerary.Trip.Stop.FareClass	Economy

Table 4.2 Parsed response as saved in the SQL parsed responses database.

CHAPTER 5. Summary and Future Work

In this thesis we proposed both an integrity model and a specification based intrusion detection system for SOA networks. The proposed Service Clark-Wilson Integrity Model (SCWIM) is a modified version of the Clark-Wilson integrity model where it incorporates the notion of a service as an integration of sub-services, service contract, concurrency and consistency, transaction sequencing and service dependencies into certification and enforcement rules of CWIM, we believe that this model can give abstraction to the SOA community for guiding the implementation and evaluation processes, and if applied to SOA can guarantee integrity, consistency and resolve concurrency problems.

Our model can be used in different ways in the future. Here is a list of possibilities for future work in this area:

- Improving weaker but more practical models of SOA security that are geared toward security evaluation.
- Developing more precise consistency models dedicated to SOA.
- Developing integrity verification and state validation tools.
- Evaluating the security of any SOA environment and pointing out the problems and enhancements that can take place.

The SOA specification based intrusion detection system is an intrusion detection system (IDS) that learns the set of behaviors and characteristics of the services in a SOA network that use SOAP messages to communicate. These behaviors and characteristics are learned from a sane data set collected in a controlled environment, where a set of tests and functions are applied to this data to extract and learn the associated behaviors and characteristics from it. A database is created to hold all of the learned characteristics to use it later in the comparison process that will determine the normal behavior from

the abnormal one to try and detect any possible attack that might take place. Our proposed specification based IDS development went through several phases which are: data collection phase, specifications learning phase, detection phase and finally the evaluation phase. Several programs were written to achieve the desired functionality for each phase as we discussed in chapter 4.

Even though our proposed IDS can detect all attacks affecting the behavior of services and will give a low false positive/negative rate if accurate specifications were developed, it can not detect attacks that mimic/do not affect the service behavior such as denial of service attacks which can affect the availability of a service in a SOA network. Despite that, we still believe that our specification-based IDS will open the door for more research in this area in the years to come.

Our developed IDS is still in the process of development and testing. A wide set of possibilities for future work exists some of which are:

- The learning data saved and used until this moment is not enough since it represents one user only using the testbed. For real life we need to have a larger data set gathered that represent a larger number of users using the testbed.
- Improving the learning process and making it iterative.
- Improving the reporting and the display process of our intrusion detection system.
- Test the testbed on other types of attacks other than the XML injection attack.
- Incorporate other types or resources of data in the learning process.
- The programs we wrote were not optimized for best performance, this might be an issue for live detection. Optimizing it will be part of a future work.
- Developing a more detailed risk analysis that fits our intrusion detection system. This risk analysis should help in developing a more precise severity level for any unusual behavior.

APPENDIX A. Data Collection Phase Source Code

This appendix shows the java source code for the first phase, the data collection phase, of the intrusion detection process. Any wsmonitor code that was not modified is not listed here. The original wsmonitor code can be found on its website [30].

A.1 Creating the Database

This section lists the source code of the function that creates the necessary tables in the SQL database for the data collection phase. This function is called from the main function in wsmonitor-collect.

```
public static boolean CreateDBTables(Statement stmt) {
   boolean DBExists = false;
   int ParametersCountMaxLimit = 20;
   String CreateRequestVarsCommand =
      "CREATE TABLE SoapIDSRequestVarsTable (
      ParseTime BIGINT PRIMARY KEY, RequestId BIGINT, VarName TEXT,
      VarType TEXT, VarValue TEXT)";
   String CreateResponseVarsCommand =
      "CREATE TABLE SoapIDSResponseVarsTable (
      ParseTime BIGINT PRIMARY KEY, ResponseId BIGINT, VarName TEXT,
      VarType TEXT, VarValue TEXT)";
   String CreateRequestTableCommand =
         "CREATE TABLE SoapIDSRequestTable (" +
         "RequestId BIGINT PRIMARY KEY," + : temporary: set to time
         "RequestingClientIP TINYTEXT," +
         "RequestingClientSourcePort SMALLINT UNSIGNED," +
         "RequestHTTPHeader TEXT," +
         "RequestSOAPMessage TEXT," +
         "RequestTime DATETIME," +
         "RequestHasAttachment boolean," +
         //temporary: set to always false
         "RequestLength INT UNSIGNED," +
```

```
        "RequestEncoding TINYTEXT";
String CreateResponseTableCommand =
        "CREATE TABLE SoapIDSResponseTable (" +
        "ResponseId BIGINT PRIMARY KEY," +  temporary: set to time
        "ResponseClientToIP TINYTEXT," + :Set same as request
        "ResponseClientToPort SMALLINT UNSIGNED," +
        :Set same as request
        "ResponseHTTPHeader TEXT," +
        "ResponseSOAPMessage TEXT," +
        "ResponseTime DATETIME," +
        "ResponseHasAttachment boolean," +
        "ResponseLength INT UNSIGNED," +
        "ResponseEncoding TINYTEXT";
String RequestTableParametersCreationCommand = "";
for (int i = 1; i <= ParametersCountMaxLimit; i = i + 1) {
   String str = "," +
         "RequestParameterName" + String.valueOf(i) + " TEXT," +
         "RequestParameterType" + String.valueOf(i) + " TEXT," +
         "RequestParameterValue" + String.valueOf(i) + " TEXT";
   RequestTableParametersCreationCommand =
      RequestTableParametersCreationCommand + str;
}
CreateRequestTableCommand = CreateRequestTableCommand + ")";
CreateResponseTableCommand = CreateResponseTableCommand + ")";
try {
   ResultSet rs1 = stmt.executeQuery("show databases");
   while (rs1.next()) {
      String s = rs1.getString(1);
      if (s.equals("LearningPhaseDataDB")) {
         DBExists = true;
      }
      System.out.println(s);
   }
} catch (SQLException sQLException) {
   sQLException.printStackTrace();
}
try {
   if (DBExists == false) {
      stmt.executeUpdate("CREATE DATABASE LearningPhaseDataDB");
   }
} catch (SQLException sQLException) {
   sQLException.printStackTrace();
}
```

```
        boolean Table1Exists = false;
        boolean Table2Exists = false;
        boolean Table3Exists = false;
        boolean Table4Exists = false;
        try {
            stmt.executeUpdate("use LearningPhaseDataDB");
            ResultSet rs2 = stmt.executeQuery("show tables");
            while (rs2.next()) {
                String s = rs2.getString(1);
                if (s.equals("SoapIDSRequestTable")) {
                    Table1Exists = true;
                }
                if (s.equals("SoapIDSResponseTable")) {
                    Table2Exists = true;
                }
                if (s.equals("SoapIDSRequestVarsTable")) {
                    Table3Exists = true;
                }
                if (s.equals("SoapIDSResponseVarsTable")) {
                    Table4Exists = true;
                }
                System.out.println(s);
            }
            if (Table1Exists == false) {
                stmt.executeUpdate(CreateRequestTableCommand);
            }
            if (Table2Exists == false) {
                stmt.executeUpdate(CreateResponseTableCommand);
            }
            if (Table3Exists == false) {
                stmt.executeUpdate(CreateRequestVarsCommand);
            }
            if (Table4Exists == false) {
                stmt.executeUpdate(CreateResponseVarsCommand);
            }
        } catch (SQLException sQLException) {
            sQLException.printStackTrace();
        }
        return true;
    }
```

A.2 Data Collection Code

This section lists the code used to collect the data during the first phase of the intrusion detection process (the data collection phase). Whenever a packet is received, the function 'run' is run on a separate thread for each received packet. This function 'run' is the main body of an object of type 'Thread' in java. The listed code for this function is a modified version of the original wsmonitor code.

```
public void run() {
   try {
      Statement resstmt = connection.createStatement();
      Statement reqstmt = connection.createStatement();
      metadata.setTime(new Date());
      String RequestIdStr = String.valueOf((
            new Date()).getTime());
      String RequestTime = getDateTime();
      //prepare the streams from host
      InputStream fromHost = socket.getInputStream();
      OutputStream toHost = socket.getOutputStream();
      String RequestingClientIP =
         socket.getInetAddress().getHostAddress();
      int RequestingClientSourcePort = socket.getPort();
      String ResponseIdStr = RequestIdStr;
      boolean RequestHasAttachment = false;
      boolean ResponseHasAttachment = false;
      // process request headers from "host"
      String requestHeaders = processRequestHeaders(fromHost);
      String RequestSOAPMessage = "";
      metadata.setRequestHeader(requestHeaders);
      // process request body from "host"
      byte[] requestMessage = processRequestBody(fromHost);
      metadata.setRequestBody(requestMessage);
      connViewer.updateRequest(metadata);
      int RequestLength = requestMessage.length;
      String RequestEncoding = null;
      String ResponseClientToIP = "";
      int ResponseClientToPort = RequestingClientSourcePort;
      java.util.Map ResponseHTTPHeader;
      String ResponseTime = "";
      String ResponseSOAPMessage = "";
      int ResponseLength = 0;
      String ResponseEncoding = null;
      HttpURLConnection targetServer;
```

```
try {
   URL url = new URL("http", connConfig.getTargetHost(),
      Integer.parseInt(connConfig.getTargetPort()), fileName);
   targetServer = (HttpURLConnection) url.openConnection();
   targetServer.setRequestMethod(methodName);
   targetServer.setDoInput(true);
   // populate headers from "host" to "target"
   Enumeration headerEnum = headersTable.keys();
   while (headerEnum.hasMoreElements()) {
      String header = (String) headerEnum.nextElement();
      targetServer.setRequestProperty(header,
         headersTable.get(header));
   }
   if (methodName.contains("POST")) {
      // open the output stream only for POST
      try {
         targetServer.setDoOutput(true);
         // write request to "target"
         OutputStream toTarget =
            targetServer.getOutputStream();
         toTarget.write(requestMessage);
         toTarget.flush();
         toTarget.close();
      } catch (Exception iOException) {
         iOException.printStackTrace();
      }
      try {
         String str;
         if (RequestEncoding != null) {
            str = new String(requestMessage, RequestEncoding);
         }
         str = new String(requestMessage, "UTF-8");
         if (str.startsWith("<?xml")) {
            RequestSOAPMessage = str;
            str = str.replace("&gt;", ">");
            str = str.replace("&lt;", "<");
            RequestSOAPMessage = str;
            java.util.Calendar calnow =
               Calendar.getInstance();
            String filename =
               String.valueOf(calnow.getTimeInMillis());
            filename = "/home/soa/XMLTest/" + "request-" +
               filename + ".xml";
```

```
                (new File("/home/soa/XMLTest/")).mkdirs();
                FileWriter fw = new FileWriter(filename);
                fw.write(str);
                fw.close();
                DocumentBuilderFactory dbfac =
                   DocumentBuilderFactory.newInstance();
                DocumentBuilder docb =
                   dbfac.newDocumentBuilder();
                org.w3c.dom.Document xmldoc =
                   docb.parse(new String(filename));
                AnalyzeSOAPRequest(xmldoc, reqstmt, RequestIdStr);
                RequestEncoding = targetServer.getContentEncoding();
                String colsStr =
                   "(RequestId,RequestingClientIP,
                   RequestingClientSourcePort,RequestHTTPHeader,
                   RequestSOAPMessage,RequestTime,
                   RequestHasAttachment,RequestLength,
                   RequestEncoding)";
                String valsStr = "('" + RequestIdStr + "','" +
                   RequestingClientIP + "','" +
                   Integer.toString(RequestingClientSourcePort) +
                   "','" + requestHeaders.replace("'", "''") +
                   "','" + RequestSOAPMessage.replace("'", "''")
                   + "','" + RequestTime;
                valsStr = valsStr + "','" + ConvertBoolToString(
                   RequestHasAttachment) + "','" +
                   Integer.toString(RequestLength) + "','" +
                   RequestEncoding + "')";
                String reqstr = "INSERT INTO SoapIDSRequestTable " +
                   colsStr + " VALUES " + valsStr;
                reqstmt.executeUpdate(reqstr);
             }
          } catch (Exception e) {
             e.printStackTrace();
          }
       }
       // check for HTTP response code
       boolean isFailure = checkResponseCode(targetServer);
       // process headers from "target"
       String responseHeader =
          processResponseHeaders(targetServer);
       metadata.setResponseHeader(responseHeader);
       // write response header to "host"
```

```
toHost.write(responseHeader.concat("\n").getBytes());
// process response body from "target"
InputStream is = isFailure ? targetServer.getErrorStream() :
   targetServer.getInputStream();
if (is != null) {
   byte[] responseBuffer = processResponseBody(is);
   ResponseLength = responseBuffer.length;
   try {
      metadata.setResponseBody(responseBuffer);
      toHost.write(responseBuffer);
   } catch (Exception iOException) {
      iOException.printStackTrace();
   }
   try {
      ResponseEncoding =
         targetServer.getContentEncoding();
      String str;
      if (ResponseEncoding != null) {
         str = new String(responseBuffer, ResponseEncoding);
      } else {
         str = new String(responseBuffer, "UTF-8");
      }
      ResponseHTTPHeader = targetServer.getHeaderFields();
      if (str.startsWith("<?xml")) {
         ResponseTime = getDateTime();
         str = str.replace("&gt;", ">");
         str = str.replace("&lt;", "<");
         ResponseClientToIP =
            socket.getInetAddress().getHostAddress();
         ResponseSOAPMessage = str;
         //ResponseId = String.valueOf((new Date()).getTime());
         java.util.Calendar calnow =
            Calendar.getInstance();
         String filename =
            String.valueOf(calnow.getTimeInMillis());
         filename = "/home/soa/XMLTest/" + "response-" +
            filename + ".xml";
         (new File("/home/soa/XMLTest/")).mkdirs();
         FileWriter fw = new FileWriter(filename);
         fw.write(str);
         fw.close();
         DocumentBuilderFactory dbfac =
            DocumentBuilderFactory.newInstance();
```

```
                DocumentBuilder docb =
                    dbfac.newDocumentBuilder();
                org.w3c.dom.Document xmldoc =
                    docb.parse(filename);
                AnalyzeSOAPResponse(xmldoc, resstmt,
                    RequestIdStr);
                String strResponseHTTPHeader =
                    processResponseHeaders(targetServer);
                String colsStr = "(ResponseId,
                    ResponseClientToIP,ResponseClientToPort,
                    ResponseHTTPHeader,ResponseSOAPMessage,
                    ResponseTime,ResponseHasAttachment,
                    ResponseLength,ResponseEncoding)";
                String valsStr = "('" + ResponseIdStr + "','" +
                    ResponseClientToIP + "','" +
                    Integer.toString(ResponseClientToPort) + "','" +
                    strResponseHTTPHeader.replace("'", "''") +
                    "','" + ResponseSOAPMessage.replace("'",
                    "''") + "','" + ResponseTime;
                valsStr = valsStr + "','" + ConvertBoolToString(
                    ResponseHasAttachment) + "','" +
                    Integer.toString(ResponseLength) + "','" +
                    ResponseEncoding + "')";
                String resstr = "INSERT INTO SoapIDSResponseTable
                    " + colsStr + " VALUES " + valsStr;
                resstmt.executeUpdate(resstr);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    toHost.flush();
    toHost.close();
} catch (UnknownHostException e) {
    metadata.setResponseBody(e.getMessage().getBytes());
    e.printStackTrace();
} catch (ConnectException e) {
    metadata.setResponseBody(e.getMessage().getBytes());
    e.printStackTrace();
} catch (IOException e) {
    metadata.setResponseBody(e.getMessage().getBytes());
    e.printStackTrace();
} catch (SecurityException e) {
```

```
            metadata.setResponseBody(e.getMessage().getBytes());
            e.printStackTrace();
        }
    } catch (Throwable t) {
        t.printStackTrace();
    } finally {
        connViewer.updateResponse(metadata);
        try {
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

A.3 XML Parse Result Class

The result of the parsing process is saved in an object of type ParametersList. Here is the definition of ParametersList:

```
static class ParametersList {

    ArrayList<String> ParametersListName;
    ArrayList<String> ParametersListType;
    ArrayList<String> ParametersListValue;
    ArrayList<String> ParametersListParseTime;

    public ParametersList() {
        ParametersListName = new ArrayList<String>();
        ParametersListType = new ArrayList<String>();
        ParametersListValue = new ArrayList<String>();
        ParametersListParseTime = new ArrayList<String>();
    }
}
```

A.4 Saving the Parsing Process Result to a Database

The parsing process of requests as well as responses is saved to a database for easier process later. The code that performs this task is listed here.

```
static void SaveRequestParametersToDB(
    ParametersList parametersList, Statement reqstmt, String Id) {
    try {
        reqstmt.executeUpdate("use LearningPhaseDataDB");
    } catch (SQLException ex) {
    }
    for (int i = 0; i < parametersList.ParametersListName.size(); i++) {
        java.util.Calendar calnow = Calendar.getInstance();
        String str0 = String.valueOf(calnow.getTimeInMillis() + i);
        //String str0 = parametersList.ParametersListParseTime.get(i);
        String str1 = parametersList.ParametersListName.get(i);
        String str2 = parametersList.ParametersListType.get(i);
        String str3 = parametersList.ParametersListValue.get(i);
        str0 = str0.replace("\'", "\'\'");
        str1 = str1.replace("\'", "\'\'");
        str2 = str2.replace("\'", "\'\'");
        str3 = str3.replace("\'", "\'\'");
        String str = "INSERT INTO SoapIDSRequestVarsTable (ParseTime ,
            RequestId, VarName, VarType, VarValue) VALUES (\'" +
            str0 + "\',\'" + Id + "\',\'" + str1 + "\',\'" +
            str2 + "\',\'" + str3 + "\')";
        try {
            reqstmt.executeUpdate(str);
        } catch (SQLException ex) {
            System.out.println(str);
        }
    }
}

static void SaveResponseParametersToDB(ParametersList parametersList,
        Statement resstmt, String Id) {
    try {
        resstmt.executeUpdate("use LearningPhaseDataDB");
    } catch (SQLException ex) {
    }
    for (int i = 0; i < parametersList.ParametersListName.size(); i++) {
        java.util.Calendar calnow = Calendar.getInstance();
        String str0 = String.valueOf(calnow.getTimeInMillis() + i);
        String str1 = parametersList.ParametersListName.get(i);
        String str2 = parametersList.ParametersListType.get(i);
        String str3 = parametersList.ParametersListValue.get(i);
        str0 = str0.replace("\'", "\'\'");
```

```
            str1 = str1.replace("\'", "\'\'");
            str2 = str2.replace("\'", "\'\'");
            str3 = str3.replace("\'", "\'\'");
            String str = "INSERT INTO SoapIDSResponseVarsTable ( ParseTime ,
               ResponseId, VarName, VarType, VarValue) VALUES (\'" +
               str0 + "\',\'"  + Id + "\',\'" + str1 + "\',\'" + str2 +
               "\',\'" + str3 + "\')";
            try {
               resstmt.executeUpdate(str);
            } catch (SQLException ex) {
               System.out.println(str);
            }
         }
      }
```

A.5 Parsing SOAP Requests

This section lists the functions used to parse SOAP requests and save the result to SQL database.

A.5.1 Initiating the Request Parsing Process

This function initiates the request parsing process and calls the function that saves the result to the database.

```
   public static void AnalyzeSOAPRequest(org.w3c.dom.Document xmldoc,
         Statement stmt, String Id) {
      org.w3c.dom.Node SOAPEnvelope = GetSOAPMessageEnvelope(xmldoc);
      org.w3c.dom.Node SOAPHeader = GetSOAPMessageHeader(SOAPEnvelope);
      org.w3c.dom.Node SOAPBody = GetSOAPMessageBody(SOAPEnvelope);
      org.w3c.dom.Node SOAPFault = GetSOAPMessageFault(SOAPBody);
      org.w3c.dom.Node SOAPRequest = GetSOAPRequest(SOAPBody);
      ParametersList parametersList = new ParametersList();

      parametersList = GetSOAPRequestParametersList(SOAPRequest, "",
         parametersList,"");
      int L = parametersList.ParametersListName.size();
      for (int i = 0; i < L; i++) {
      }
      SaveRequestParametersToDB(parametersList, stmt, Id);
      return;
   }
```

A.5.2 Parsing Request XML Node into a List of Tags Names and Values

This function takes a handle to an XML request node and returns the list of tags and there values and types contained in that node.

```
static ParametersList GetSOAPRequestParametersList(org.w3c.dom.Node
      RequestNode, String NamePrefix, ParametersList parametersList,
      String indent) {
   indent = indent + "  ";
   org.w3c.dom.NodeList nodeList = RequestNode.getChildNodes();
   int L = nodeList.getLength();
   if (L <= 0) {
      return parametersList;
   }
   for (int i = 0; i < L; i++) {
      org.w3c.dom.Node node = nodeList.item(i);
      if (node.getChildNodes().getLength() == 0) {
         String PName = NamePrefix.substring(1);
         String PValue = node.getNodeValue();
         String PType = node.getNodeName();
         java.util.Calendar calnow = Calendar.getInstance();
         String ParseTime = String.valueOf(calnow.getTimeInMillis()+i);
         boolean cond1 = PValue.trim().equals("");
         boolean cond2 = false;
         boolean cond3 = false;
         if ((i + 1) < L) {
            org.w3c.dom.Node node2 = nodeList.item(i + 1);
            String PType2 = node2.getNodeName();
            if (!PType2.equalsIgnoreCase("#comment")) {
               cond2 = true;
            }
         }
         if ((i - 1) > 0) {
            org.w3c.dom.Node node3 = nodeList.item(i - 1);
            String PType3 = node3.getNodeName();
            if (!PType3.equalsIgnoreCase("#comment")) {
               cond3 = true;
            }
         }
         if (!((cond1 && cond2) || (cond1 && cond3))) {
            System.out.println(indent + i + "\t" + PName + "\t" +
               PType + "\t" + PValue);
            parametersList =
```

```
                AddToParametersListArray(parametersList,
                ParseTime, PName, PType, PValue);
            }
        }
        String str = NamePrefix + "." + node.getNodeName();
        parametersList = GetSOAPRequestParametersList(node,
            str, parametersList, indent);
    }
    return parametersList;
}
```

This function returns the request SOAP message enclosed in the body of request XML document.

```
static org.w3c.dom.Node GetSOAPRequest(org.w3c.dom.Node Body) {
    org.w3c.dom.NodeList RequestList = Body.getChildNodes();
    if (RequestList.getLength() <= 0) {
        return null;
    }
    if ((RequestList.getLength() == 1) &&
        (RequestList.item(0).getNodeName().endsWith(":Fault"))) {
        return null;
    }
    if ((RequestList.getLength() == 1) &&
        !(RequestList.item(0).getNodeName().endsWith(":Fault"))) {
        return RequestList.item(0);
    }
    if (RequestList.getLength() > 1) {
        return Body;
    }
    return null;
}
```

A.6 Parsing SOAP Responses

This section lists the functions used to parse SOAP responses and save the result to SQL database.

A.6.1 Initiating the Response Parsing Process

This function initiates the response parsing process and calls the function that saves the result to the database.

```
public static void AnalyzeSOAPResponse(org.w3c.dom.Document xmldoc,
      Statement stmt, String Id) {
   org.w3c.dom.Node SOAPEnvelope = GetSOAPMessageEnvelope(xmldoc);
   org.w3c.dom.Node SOAPHeader = GetSOAPMessageHeader(SOAPEnvelope);
   org.w3c.dom.Node SOAPBody = GetSOAPMessageBody(SOAPEnvelope);
   org.w3c.dom.Node SOAPFault = GetSOAPMessageFault(SOAPBody);
   org.w3c.dom.Node SOAPResponse = GetSOAPResponse(SOAPBody);
   ParametersList parametersList = new ParametersList();

   parametersList = GetSOAPResponseParametersList(SOAPResponse, "",
      parametersList, "");
   int L = parametersList.ParametersListName.size();
   for (int i = 0; i < L; i++) {
   }
   SaveResponseParametersToDB(parametersList, stmt, Id);
   return;
}
```

A.6.2 Parsing Response XML Node into a List of Tags Names and Values

This function takes a handle to an XML response node and returns the list of tags and there values and types contained in that node.

```
static ParametersList GetSOAPResponseParametersList(org.w3c.dom.Node
      ResponseNode, String NamePrefix, ParametersList
      parametersList, String indent) {
   indent = indent + "  ";
   if (ResponseNode == null) {
      return parametersList;
   }
   org.w3c.dom.NodeList nodeList = ResponseNode.getChildNodes();
   int L = nodeList.getLength();
   if (L <= 0) {
      return parametersList;
   }
   for (int i = 0; i < L; i++) {
      org.w3c.dom.Node node = nodeList.item(i);
      if (node.getChildNodes().getLength() == 0) {
         String PName;
```

```
            if (NamePrefix.length() > 1) {
               PName = NamePrefix.substring(1);
            } else {
               PName = null;
            }
            String PValue = node.getNodeValue();
            String PType = node.getNodeName();
            java.util.Calendar calnow = Calendar.getInstance();
            String ParseTime = String.valueOf(calnow.getTimeInMillis()
               + i);
            boolean cond1 = PValue.trim().equals("");
            boolean cond2 = false;
            boolean cond3 = false;
            if ((i + 1) < L) {
               org.w3c.dom.Node node2 = nodeList.item(i + 1);
               String PType2 = node2.getNodeName();
               if (!PType2.equalsIgnoreCase("#comment")) {
                  cond2 = true;
               }
            }
            if ((i - 1) > 0) {
               org.w3c.dom.Node node3 = nodeList.item(i - 1);
               String PType3 = node3.getNodeName();
               if (!PType3.equalsIgnoreCase("#comment")) {
                  cond3 = true;
               }
            }
            if (!((cond1 && cond2) || (cond1 && cond3))) {
               System.out.println(indent + i + "\t" + PName +
                  "\t" + PType + "\t" + PValue);
               parametersList =
                  AddToParametersListArray(parametersList,
                  ParseTime, PName, PType, PValue);
            }
         }
         String str = NamePrefix + "." + node.getNodeName();
         parametersList = GetSOAPResponseParametersList(node,
            str, parametersList, indent);
      }
      return parametersList;
   }
```

This function returns the response SOAP message enclosed in the body of request XML document.

```
static org.w3c.dom.Node GetSOAPResponse(org.w3c.dom.Node Body) {
   if (Body == null) {
      return null;
   }
   org.w3c.dom.NodeList ResponseList = Body.getChildNodes();
   if (ResponseList.getLength() <= 0) {
      return null;
   }
   if ((ResponseList.getLength() == 1) &&
      (ResponseList.item(0).getNodeName().endsWith(":Fault"))) {
      return null;
   }
   if ((ResponseList.getLength() == 1) &&
      !(ResponseList.item(0).getNodeName().endsWith(":Fault"))) {
      return ResponseList.item(0);
   }
   if (ResponseList.getLength() > 1) {
      return Body;
   }
   return null;
}
```

A.7 Common Functions Used to by the Request and Response Parsing Process

This function returns a handle to the envelope of the SOAP message.

```
static org.w3c.dom.Node GetSOAPMessageEnvelope(org.w3c.dom.Document
      xmldoc) {
   org.w3c.dom.Node Envelope = xmldoc.getDocumentElement();
   if (Envelope.getNodeName().endsWith(":Envelope") == true) {
      return Envelope;
   } else {
      return null;
   }
}
```

This function returns a handle to the header of the SOAP message.

```
static org.w3c.dom.Node GetSOAPMessageHeader(org.w3c.dom.Node
    Envelope) {
  if (Envelope == null) {
    return null;
  }
  org.w3c.dom.NodeList EnvelopeNodes = Envelope.getChildNodes();
  int NodesCount = EnvelopeNodes.getLength();
  for (int i = 0; i < NodesCount; i++) {
    if (EnvelopeNodes.item(i).getNodeName().endsWith(":Header") ==
      true) {
      return EnvelopeNodes.item(i);
    }
  }
  return null;
}
```

This function returns a handle to the body of a SOAP message.

```
static org.w3c.dom.Node GetSOAPMessageBody(org.w3c.dom.Node Envelope) {
  if (Envelope == null) {
    return null;
  }
  org.w3c.dom.NodeList EnvelopeNodes = Envelope.getChildNodes();
  int NodesCount = EnvelopeNodes.getLength();
  for (int i = 0; i < NodesCount; i++) {
    if (EnvelopeNodes.item(i).getNodeName().endsWith(":Body") ==
      true) {
      return EnvelopeNodes.item(i);
    }
  }
  return null;
}
```

This functions returns a handle to the fault part of a SOAP message.

```
static org.w3c.dom.Node GetSOAPMessageFault(org.w3c.dom.Node Body) {
  if (Body == null) {
    return null;
  }
  org.w3c.dom.NodeList BodyNodes = Body.getChildNodes();
  int NodesCount = BodyNodes.getLength();
  for (int i = 0; i < NodesCount; i++) {
    if (BodyNodes.item(i).getNodeName().endsWith(":Fault") == true){
```

```
            return BodyNodes.item(i);
        }
    }
    return null;
}
```

This function is used to loop through the nodes and sub-nodes of XML document during the parsing process.

```
static ParametersList AddToParametersListArray(ParametersList OldList,
        String ParseTime , String PName, String PType, String PValue) {
    try {
        if (PName != null) {
            OldList.ParametersListName.add(PName);
        } else {
            OldList.ParametersListName.add("");
        }
        if (PType != null) {
            OldList.ParametersListType.add(PType);
        } else {
            OldList.ParametersListType.add("");
        }
        if (PValue != null) {
            OldList.ParametersListValue.add(PValue);
        } else {
            OldList.ParametersListValue.add("");
        }
        if (ParseTime != null) {
            OldList.ParametersListParseTime.add(ParseTime);
        } else {
            OldList.ParametersListParseTime.add("");
        }
        return OldList;
    } catch (Throwable t) {
        t.printStackTrace();
    }
    return null;
}
```

This function clears ParametersList object.

```
static ParametersList CleanUpParameters(ParametersList
    parametersList)
{
    ParametersList retval = new ParametersList();
    int L = parametersList.ParametersListName.size();
    for (int i = 0; i < L; i++)
    {
        if (!parametersList.ParametersListName.get(i).
            trim().equals(""))
        {
            retval.ParametersListName.add(
                parametersList.ParametersListName.get(i));
            retval.ParametersListType.add(
                parametersList.ParametersListType.get(i));
            retval.ParametersListValue.add(
                parametersList.ParametersListValue.get(i));
            retval.ParametersListParseTime.add(
                parametersList.ParametersListParseTime.get(i));
        }
    }
    return retval;
}
```

APPENDIX B. Learning Phase Source Code

This appendix shows the java source code for the second phase, the learning phase, of the intrusion detection process.

B.1 Main Function

This is the main function that initiates the learning process.

```
public class Main {

   /**
    * @param args the command line arguments
    */
   public static void main(String[] args) {
         // This main routine calls all subroutines that learn
         //the various characteristics of the collected data
         Connection con;
         // This is the address of the database on which the clean
         //data is saved
         // We want to learn the characteristics of the sane data
         //on this database LearningPhaseDataDB
      String gdsadd = "jdbc:mysql://vm-gds:3306/
         LearningPhaseDataDB?holdResultsOpenOverStatementClose=true";
         String gdsusr = "gdsuser";
         String gdspwd = "gdsDBpassword";
         //This is the address of the database on which the
         //result of the learning process will be saved
      String gdsLearnDB = "jdbc:mysql://vm-gds:3306/
         LearningDB?holdResultsOpenOverStatementClose=true";
      try {
         Class.forName("com.mysql.jdbc.Driver");
         Class.forName("com.mysql.jdbc.Driver").newInstance();
         //Debug only: debug the process of initializing
```

```
//the jdbc java driver
con = DriverManager.getConnection(gdsadd, gdsusr, gdspwd);
//connect to the databse on which the collection
//process data is saved
String DBName = "LearningPhaseDataDB";
String ReqVarTable = "SoapIDSRequestVarsTable";
//Collection process request data variables table
String ResVarTable = "SoapIDSResponseVarsTable";
//Collection process response data variables table
String ReqTable = "SoapIDSRequestTable";
//Collection process request table for other information :
//time, headers, length, id, encoding ... etc
String ResTable = "SoapIDSResponseTable";
//Collection process response table for other information :
//time, headers, length, id, encoding ... etc
String VarNameCol = "VarName";
// name of the column that contains the variables names in
//SoapIDSRequestVarsTable and SoapIDSResponseVarsTable
String VarValueCol = "VarValue";
// name of the column that contains the variables values in
//SoapIDSRequestVarsTable and SoapIDSResponseVarsTable
String CharSet = "%><";
String RequestLengthCol = "RequestLength";
// name of the column that contains the request message
//length in SoapIDSRequestTable
String ResponseLengthCol = "ResponseLength";
// name of the column that contains the response message
//length in SoapIDSResponseTable
String ReqEncodingCol = "RequestEncoding";
// name of the column that contains the request message
//encoding in SoapIDSRequestTable
String ResEncodingCol = "ResponseEncoding";
// name of the column that contains the response message
//encoding in SoapIDSResponseTable
//learn the length of every xml tag in every soap
//message for request data and response data
//result saved in a two dimensional array
String[][] ReqDataLangth = LearnDataLength(con, DBName,
   ReqVarTable, VarNameCol, VarValueCol);
String[][] ResDataLangth = LearnDataLength(con, DBName,
   ResVarTable, VarNameCol, VarValueCol);
//LearnDataLength: 3xL: variable name, min value, max value
// L is the number of unique xml tags names. this number
```

```
//is found by counting unique names of xml tags in VarNameCol
//Learn if a variable value is boolean or not
//result saved in a two dimensional array
String[][] ReqDataIsBool = LearnCastBool(con, DBName,
   ReqVarTable, VarNameCol, VarValueCol);
String[][] ResDataIsBool = LearnCastBool(con, DBName,
   ResVarTable, VarNameCol, VarValueCol);
//LearnCastBool: 2xL:
//variable name, boolean=always bool? (true,false)
// L is the number of unique xml tags names. this number
//is found by counting unique names of xml tags in VarNameCol
//learn if the xml tag variable value is a date-time or not
//result saved in a two dimensional array
String[][] ReqDataIsDate = LearnCastDate(con, DBName,
   ReqVarTable, VarNameCol, VarValueCol);
String[][] ResDataIsDate = LearnCastDate(con, DBName,
   ResVarTable, VarNameCol, VarValueCol);
//ReqDataIsDate: 2xL:
//variable name, boolean=always date? (true,false)
// L is the number of unique xml tags names. this number is
//found by counting unique names of xml tags in VarNameCol
//Learn if an xml tag can be casted to number or not
//result saved in a two dimensional array
String[][] ReqDataIsNum = LearnCastNum(con, DBName,
   ReqVarTable, VarNameCol, VarValueCol);
String[][] ResDataIsNum = LearnCastNum(con, DBName,
   ResVarTable, VarNameCol, VarValueCol);
//LearnCastNum: 2xL:
//variable name, boolean=always number? (true,false)
// L is the number of unique xml tags names. this number is
//found by counting unique names of xml tags in VarNameCol
//Learn if an xml tag value has one of the characters
//in CharSet (such as "%<>" etc)
//if at least one of those characters is present
//it will return true
//if none of those characters is present it will return false
//result saved in a two dimensional array
String[][] ReqDataHasChar = LearnHasChar(con, DBName,
   ReqVarTable, VarNameCol, VarValueCol, CharSet);
String[][] ResDataHasChar = LearnHasChar(con, DBName,
   ResVarTable, VarNameCol, VarValueCol, CharSet);
//LearnHasChar: 2xL:
   variable name, boolean=always given char? (true,false)
```

```
// L is the number of unique xml tags names. this number is
//found by counting unique names of xml tags in VarNameCol
//learn the minimum and maximum length of
//all request soap messages
//ReqSOAPLen[0] contains the minimum detected length
//ReqSOAPLen[1] contains the maximum detected length
int[] ReqSOAPLen = LearnSOAPMessageLength(con, DBName,
   ReqTable, RequestLengthCol);
//learn the minimum and maximum length of all
//response soap messages
int[] ResSOAPLen = LearnSOAPMessageLength(con, DBName,
   ResTable, ResponseLengthCol);
//Learn the minimum and maximum length of SOAP
//requests and responses
//return only an int pair for each LearnSOAPMessageLength call
//
//learn the encoding type of all soap messages
//for requests and responses
//result saved in a two dimensional array
String[] ReqEncodingList = LearnEncoding(con, DBName,
   ReqTable, ReqEncodingCol);
String[] ResEncodingList = LearnEncoding(con, DBName,
   ResTable, ResEncodingCol);
//returns a list of all possible encodings for requests/responses
//learn the count of each xml (minimum and maximum)
//tag in all soap messages for requests and responses
//result saved in a two dimensional array
String[][] RequestNameCountRange = LearnTagsCountRange(con,
   DBName, ReqVarTable, VarNameCol, VarValueCol, "RequestId");
String[][] ResponseNamesCountRange = LearnTagsCountRange(con,
   DBName, ResVarTable, VarNameCol, VarValueCol, "ResponseId");
//retval = new String[3][LAllUniqNames];
//retval[0]: var name
//retval[1]: minimum count in a doc if it appears
//retval[2]: maximum count in a doc if it appears
//learn the calls sequence of all xml tags
//more comments about this call inside the function itself
Object[] SequenceTables = LearnCallsSequence(con, DBName,
   ReqVarTable, ResVarTable, VarNameCol, VarValueCol,
   VarNameCol, VarValueCol);
//returns an array of four elements. Each element is a 2D array
//SequenceTables[0] = res_always_preceded_by_req;
//SequenceTables[1] = res_may_preceded_by_req;
```

```
        //SequenceTables[2] = NORTable; //not used
        //SequenceTables[3] = NANDTable; //not used
        //boolean res_always_preceded_by_req[][] =
        //new boolean[LenRequestVarName][LenResponseVarName];
        //boolean res_may_preceded_by_req[][] =
        //new boolean[LenRequestVarName][LenResponseVarName];
        //boolean NORTable[][] = new boolean[LenRequestVarName]
        //[LenResponseVarName]; //not used
        //boolean NANDTable[][] = new boolean[LenRequestVarName]
        //[LenResponseVarName]; //not used
        //Save the results of the learning process to gdsLearnDB
        //"jdbc:mysql://vm-gds:3306/LearningDB?
        //holdResultsOpenOverStatementClose=true"
        DumpDataToLearningDB(gdsLearnDB, gdsusr, gdspwd,
            ReqDataLangth, ResDataLangth, ReqDataIsBool,
            ResDataIsBool, ReqDataIsDate, ResDataIsDate,
            ReqDataIsNum, ResDataIsNum, ReqDataHasChar, ResDataHasChar,
            ReqSOAPLen, ResSOAPLen, ReqEncodingList, ResEncodingList,
            RequestNameCountRange, ResponseNamesCountRange,
            SequenceTables);

    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

B.2 Saving the Result to the DataBase

This function saves the result of the learning process to an SQL database.

```
static void DumpDataToLearningDB(String dburl, String gdsusr,
    String gdspwd, String[][] ReqDataLength,
    String[][] ResDataLength, String[][] ReqDataIsBool,
    String[][] ResDataIsBool,
    String[][] ReqDataIsDate, String[][] ResDataIsDate,
    String[][] ReqDataIsNum, String[][] ResDataIsNum,
    String[][] ReqDataHasChar, String[][] ResDataHasChar,
    int[] ReqSOAPLen, int[] ResSOAPLen,
    String[] ReqEncodingList, String[] ResEncodingList,
    String[][] RequestNameCountRange,
    String[][] ResponseNamesCountRange,
```

```
Object[] SequenceTables) {
//This routine saves the results of the
//learning process to a database
try {
   Class.forName("com.mysql.jdbc.Driver");
   Class.forName("com.mysql.jdbc.Driver").newInstance();
   //debug purposes only to check driver is installed properly
   Connection con = DriverManager.getConnection(dburl,
      gdsusr, gdspwd);
   //connect to the database on which the result will be saved
   String DBName = "LearningDB";
   Statement stmt1 = con.createStatement();
   Statement stmt2 = con.createStatement();
   Statement stmt3 = con.createStatement();
   Statement stmt4 = con.createStatement();
   stmt1.executeUpdate("use " + DBName);
   stmt2.executeUpdate("use " + DBName);
   stmt3.executeUpdate("use " + DBName);
   stmt4.executeUpdate("use " + DBName);
   Object ANDTableO = SequenceTables[0];
   Object ORTableO = SequenceTables[1];
   Object NORTableO = SequenceTables[2];
   Object NANDTableO = SequenceTables[3];
   Object ReqNamesO = SequenceTables[4];
   Object ResNamesO = SequenceTables[5];
   boolean ANDTable[][] = (boolean[][]) ANDTableO;
   //cast Object to boolean 2D array
   boolean ORTable[][] = (boolean[][]) ORTableO;
   //cast Object to boolean 2D array
   boolean NORTable[][] = (boolean[][]) NORTableO;
   //cast Object to boolean 2D array
   boolean NANDTable[][] = (boolean[][]) NANDTableO;
   //cast Object to boolean 2D array
   String ReqNames[] = (String[]) ReqNamesO;
   //cast Object to 1D string array
   String ResNames[] = (String[]) ResNamesO;
   //cast Object to 1D string array
   int LenReqVarName1 = ANDTable.length;
   //read length of boolean table
   int LenResVarName1 = ANDTable[0].length;
   int LenReqVarName2 = ORTable.length;
   //read length of boolean table
   int LenResVarName2 = ORTable[0].length;
```

```
int LenReqVarName3 = NORTable.length;
//read length of boolean table
int LenResVarName3 = NORTable[0].length;
int LenReqVarName4 = NANDTable.length;
//read length of boolean table
int LenResVarName4 = NANDTable[0].length;
//boolean ANDTable[][] = new
//boolean[LenRequestVarName][LenResponseVarName];
//boolean ORTable[][] = new
//boolean[LenRequestVarName][LenResponseVarName];
//boolean NORTable[][] = new
//boolean[LenRequestVarName][LenResponseVarName];
//boolean NANDTable[][] = new
//boolean[LenRequestVarName][LenResponseVarName];
String Cols = "";
for (int i = 0; i < (LenResVarName4 - 1); i++) {
   Cols = Cols + "Res" + String.valueOf(i) + " TINYINT(1),";
}
Cols = Cols + "Res" + String.valueOf(LenResVarName4 - 1)
   + " TINYINT(1)";
//Recreate all tables in the database
//CallsSequenceAND
try {
   try {
      stmt1.executeUpdate("DROP TABLE CallsSequenceAND");
   } catch (SQLException sQLException) {
      sQLException.printStackTrace();
   }
   stmt1.executeUpdate("CREATE TABLE CallsSequenceAND ("
   + Cols + ")");
} catch (Exception e) {
   e.printStackTrace();
}
//recreate CallsSequenceNAND table
try {
   try {
      stmt1.executeUpdate("DROP TABLE CallsSequenceNAND");
   } catch (SQLException sQLException) {
      sQLException.printStackTrace();
   }
   stmt1.executeUpdate("CREATE TABLE CallsSequenceNAND ("
   + Cols + ")");
} catch (Exception e) {
```

```
            e.printStackTrace();
        }
        //recreate CallsSequenceOR table
        try {
            try {
                stmt1.executeUpdate("DROP TABLE CallsSequenceOR");
            } catch (SQLException sQLException) {
                sQLException.printStackTrace();
            }
            stmt1.executeUpdate("CREATE TABLE CallsSequenceOR ("
            + Cols + ")");
        } catch (Exception e) {
            e.printStackTrace();
        }
        //recreate CallsSequenceNOR table
        try {
            try {
                stmt1.executeUpdate("DROP TABLE CallsSequenceNOR");
            } catch (SQLException sQLException) {
                sQLException.printStackTrace();
            }
            stmt1.executeUpdate("CREATE TABLE CallsSequenceNOR ("
            + Cols + ")");
        } catch (Exception e) {
            e.printStackTrace();
        }
        //recreate ReqTagsNames,ResTagsNames table
        try {
            stmt1.executeUpdate("DROP TABLE ReqTagsNames");
        } catch (SQLException sQLException) {
            sQLException.printStackTrace();
        }
        try {
            stmt1.executeUpdate("CREATE TABLE ReqTagsNames (Idx INTEGER ,
                ReqTag TEXT)");
        } catch (SQLException sQLException) {
            sQLException.printStackTrace();
        }
        try {
            stmt1.executeUpdate("DROP TABLE ResTagsNames");
        } catch (SQLException sQLException) {
            sQLException.printStackTrace();
        }
```

```
try {
    stmt1.executeUpdate("CREATE TABLE ResTagsNames (Idx INTEGER,
        ResTag TEXT)");
} catch (SQLException sQLException) {
    sQLException.printStackTrace();
}
// Start saving data in the database
for (int i = 0; i < LenReqVarName3; i++) {
    try {
        stmt1.executeUpdate("INSERT INTO ReqTagsNames VALUES(\'"
            + i + "\',\'" + ReqNames[i] + "\')");
    } catch (SQLException sQLException) {
        sQLException.printStackTrace();
    }
}
for (int i = 0; i < LenResVarName3; i++) {
    try {
        stmt1.executeUpdate("INSERT INTO ResTagsNames VALUES(\'"
            + i + "\',\'" + ResNames[i] + "\')");
    } catch (SQLException sQLException) {
        sQLException.printStackTrace();
    }
}
// start saving CallsSequenceAND table
for (int i = 0; i < LenReqVarName4; i++) {
    String str = "INSERT INTO CallsSequenceAND VALUES (\'";
    for (int j = 0; j < (LenResVarName2 - 1); j++) {
        int k = 0;
        if (ANDTable[i][j] == true) {
            k = 1;
        }
        if (ANDTable[i][j] == false) {
            k = 0;
        }
        str = str + k + "\',\'";
    }
    int m = 0;
    if (ANDTable[i][LenResVarName2 - 1] == true) {
        m = 1;
    }
    if (ANDTable[i][LenResVarName2 - 1] == false) {
        m = 0;
    }
```

```
            str = str + String.valueOf(m) + "\')";
            stmt1.executeUpdate(str);
        }
        //start saving CallsSequenceOR
        for (int i = 0; i < LenReqVarName4; i++) {
            String str = "INSERT INTO CallsSequenceOR VALUES (\'";
            for (int j = 0; j < (LenResVarName2 - 1); j++) {
                int k = 0;
                if (ORTable[i][j] == true) {
                    k = 1;
                }
                if (ORTable[i][j] == false) {
                    k = 0;
                }
                str = str + k + "\',\'";
            }
            int m = 0;
            if (ORTable[i][LenResVarName2 - 1] == true) {
                m = 1;
            }
            if (ORTable[i][LenResVarName2 - 1] == false) {
                m = 0;
            }
            str = str + String.valueOf(m) + "\')";
            stmt1.executeUpdate(str);
        }
        //start saving CallsSequenceNOR
        for (int i = 0; i < LenReqVarName4; i++) {
            String str = "INSERT INTO CallsSequenceNOR VALUES (\'";
            for (int j = 0; j < (LenResVarName2 - 1); j++) {
                int k = 0;
                if (NORTable[i][j] == true) {
                    k = 1;
                }
                if (NORTable[i][j] == false) {
                    k = 0;
                }
                str = str + k + "\',\'";
            }
            int m = 0;
            if (NORTable[i][LenResVarName2 - 1] == true) {
                m = 1;
            }
```

```
            if (NORTable[i][LenResVarName2 - 1] == false) {
                m = 0;
            }
            str = str + String.valueOf(m) + "\')";
            stmt1.executeUpdate(str);
        }
        //start saving CallsSequenceNAND table
        for (int i = 0; i < LenReqVarName4; i++) {
            String str = "INSERT INTO CallsSequenceNAND VALUES (\'";
            for (int j = 0; j < (LenResVarName2 - 1); j++) {
                int k = 0;
                if (NANDTable[i][j] == true) {
                    k = 1;
                }
                if (NANDTable[i][j] == false) {
                    k = 0;
                }
                str = str + k + "\',\'";
            }
            int m = 0;
            if (NANDTable[i][LenResVarName2 - 1] == true) {
                m = 1;
            }
            if (NANDTable[i][LenResVarName2 - 1] == false) {
                m = 0;
            }
            str = str + String.valueOf(m) + "\')";
            stmt1.executeUpdate(str);
        }
        //stmt1.executeUpdate(Cols);
        try {
            try {
                stmt1.executeUpdate("DROP TABLE ReqDataLength");
            } catch (SQLException sQLException) {
                sQLException.printStackTrace();
            }
            stmt1.executeUpdate("CREATE TABLE ReqDataLength (Name TEXT,
                Min INTEGER, Max INTEGER)");
        } catch (Exception e) {
            e.printStackTrace();
        }
        try {
            try {
```

```
            stmt1.executeUpdate("DROP TABLE ResDataLength");
        } catch (SQLException sQLException) {
            sQLException.printStackTrace();
        }
        stmt1.executeUpdate("CREATE TABLE ResDataLength (Name TEXT,
            Min INTEGER, Max INTEGER)");
    } catch (Exception e) {
        e.printStackTrace();
    }
    try {
        try {
            stmt2.executeUpdate("DROP TABLE ReqDataIsBool");
        } catch (SQLException sQLException) {
            sQLException.printStackTrace();
        }
        stmt2.executeUpdate("CREATE TABLE ReqDataIsBool (Name TEXT,
            IsBool TINYINT(1))");
    } catch (Exception e) {
        e.printStackTrace();
    }
    try {
        try {
            stmt2.executeUpdate("DROP TABLE ResDataIsBool");
        } catch (SQLException sQLException) {
            sQLException.printStackTrace();
        }
        stmt2.executeUpdate("CREATE TABLE ResDataIsBool (Name TEXT,
            IsBool TINYINT(1))");
    } catch (Exception e) {
        e.printStackTrace();
    }
    try {
        try {
            stmt2.executeUpdate("DROP TABLE ReqDataIsDate");
        } catch (SQLException sQLException) {
            sQLException.printStackTrace();
        }
        stmt2.executeUpdate("CREATE TABLE ReqDataIsDate (Name TEXT,
            IsDate TINYINT(1))");
    } catch (Exception e) {
        e.printStackTrace();
    }
    try {
```

```
            try {
                stmt2.executeUpdate("DROP TABLE ResDataIsDate");
            } catch (SQLException sQLException) {
                sQLException.printStackTrace();
            }
            stmt2.executeUpdate("CREATE TABLE ResDataIsDate (Name TEXT,
                IsDate TINYINT(1))");
        } catch (Exception e) {
            e.printStackTrace();
        }
        try {
            try {
                stmt2.executeUpdate("DROP TABLE ReqDataIsNum");
            } catch (SQLException sQLException) {
                sQLException.printStackTrace();
            }
            stmt2.executeUpdate("CREATE TABLE ReqDataIsNum (Name TEXT,
                IsNum TINYINT(1))");
        } catch (Exception e) {
            e.printStackTrace();
        }
        try {
            try {
                stmt2.executeUpdate("DROP TABLE ResDataIsNum");
            } catch (SQLException sQLException) {
                sQLException.printStackTrace();
            }
            stmt2.executeUpdate("CREATE TABLE ResDataIsNum (Name TEXT,
                IsNum TINYINT(1))");
        } catch (Exception e) {
            e.printStackTrace();
        }
        try {
            try {
                stmt2.executeUpdate("DROP TABLE ReqDataHasChar");
            } catch (SQLException sQLException) {
                sQLException.printStackTrace();
            }
            stmt2.executeUpdate("CREATE TABLE ReqDataHasChar (Name TEXT,
                HasChar TINYINT(1))");
        } catch (Exception e) {
            e.printStackTrace();
        }
```

```
        try {
            try {
                stmt2.executeUpdate("DROP TABLE ResDataHasChar");
            } catch (SQLException sQLException) {
                sQLException.printStackTrace();
            }
            stmt2.executeUpdate("CREATE TABLE ResDataHasChar (Name TEXT,
                HasChar TINYINT(1))");
        } catch (Exception e) {
            e.printStackTrace();
        }
        try {
            try {
                stmt3.executeUpdate("DROP TABLE ReqSOAPLen");
            } catch (SQLException sQLException) {
                sQLException.printStackTrace();
            }
            stmt3.executeUpdate("CREATE TABLE ReqSOAPLen (MinLen INTEGER,
                MaxLen INTEGER)");
        } catch (Exception e) {
            e.printStackTrace();
        }
        try {
            try {
                stmt3.executeUpdate("DROP TABLE ResSOAPLen");
            } catch (SQLException sQLException) {
                sQLException.printStackTrace();
            }
            stmt3.executeUpdate("CREATE TABLE ResSOAPLen (MinLen INTEGER,
                MaxLen INTEGER)");
        } catch (Exception e) {
            e.printStackTrace();
        }
        try {
            try {
                stmt3.executeUpdate("DROP TABLE ReqEncodingList");
            } catch (SQLException sQLException) {
                sQLException.printStackTrace();
            }
            stmt3.executeUpdate("CREATE TABLE ReqEncodingList (
                Encoding TEXT)");
        } catch (Exception e) {
            e.printStackTrace();
```

```
        }
        try {
            try {
                stmt3.executeUpdate("DROP TABLE ResEncodingList");
            } catch (SQLException sQLException) {
                sQLException.printStackTrace();
            }
            stmt3.executeUpdate("CREATE TABLE ResEncodingList (
                Encoding TEXT)");
        } catch (Exception e) {
            e.printStackTrace();
        }
        try {
            try {
                stmt3.executeUpdate("DROP TABLE ReqNameCountRange");
            } catch (SQLException sQLException) {
                sQLException.printStackTrace();
            }
            stmt3.executeUpdate("CREATE TABLE ReqNameCountRange (
                Name TEXT, Min INTEGER, Max INTEGER)");
        } catch (Exception e) {
            e.printStackTrace();
        }
        try {
            try {
                stmt3.executeUpdate("DROP TABLE ResNameCountRange");
            } catch (SQLException sQLException) {
                sQLException.printStackTrace();
            }
            stmt3.executeUpdate("CREATE TABLE ResNameCountRange (
                Name TEXT, Min INTEGER, Max INTEGER)");
        } catch (Exception e) {
            e.printStackTrace();
        }
        int L1 = ReqDataLength[0].length;
        int L2 = ResDataLength[0].length;
        int L3 = ReqDataIsBool[0].length;
        int L4 = ResDataIsBool[0].length;
        int L5 = ReqDataIsDate[0].length;
        int L6 = ResDataIsDate[0].length;
        int L7 = ReqDataIsNum[0].length;
        int L8 = ResDataIsNum[0].length;
        int L9 = ReqDataHasChar[0].length;
```

```
int L10 = ResDataHasChar[0].length;
int L11 = ReqEncodingList.length;
int L12 = ResEncodingList.length;
int L13 = RequestNameCountRange[0].length;
int L14 = ResponseNamesCountRange[0].length;
for (int i = 0; i < L1; i++) {
   String reqname = ReqDataLength[0][i];
   String reqmin = ReqDataLength[1][i];
   String reqmax = ReqDataLength[2][i];
   if (reqname.trim().equals("") == false) {
      String str1 = "INSERT INTO ReqDataLength VALUES (\'" +
         reqname + "\',\'" + reqmin + "\',\'" + reqmax + "\')";
      stmt1.executeUpdate(str1);
      System.out.println(str1);
   }
}
for (int i = 0; i < L2; i++) {
   String resname = ResDataLength[0][i];
   String resmin = ResDataLength[1][i];
   String resmax = ResDataLength[2][i];
   if (resname.trim().equals("") == false) {
      String str2 = "INSERT INTO ResDataLength VALUES (\'" +
         resname + "\',\'" + resmin + "\',\'" + resmax + "\')";
      stmt2.executeUpdate(str2);
      System.out.println(str2);
   }
}
for (int i = 0; i < L3; i++) {
   String reqname = ReqDataIsBool[0][i];
   Boolean isbool = Boolean.valueOf(ReqDataIsBool[1][i]);
   int intIsBool = 0;
   if (isbool) {
      intIsBool = 1;
   }
   if (!isbool) {
      intIsBool = 0;
   }
   if (reqname.trim().equals("") == false) {
      String str2 = "INSERT INTO ReqDataIsBool VALUES (\'" +
         reqname + "\',\'" + intIsBool + "\')";
      stmt3.executeUpdate(str2);
      System.out.println(str2);
   }
```

```
        }
        for (int i = 0; i < L4; i++) {
            String resname = ResDataIsBool[0][i];
            Boolean isbool = Boolean.valueOf(ResDataIsBool[1][i]);
            int intIsBool = 0;
            if (isbool) {
                intIsBool = 1;
            }
            if (!isbool) {
                intIsBool = 0;
            }
            if (resname.trim().equals("") == false) {
                String str2 = "INSERT INTO ResDataIsBool VALUES (\'" +
                    resname + "\',\'" + intIsBool + "\')";
                stmt4.executeUpdate(str2);
                System.out.println(str2);
            }
        }
        for (int i = 0; i < L5; i++) {
            String reqname = ReqDataIsDate[0][i];
            Boolean isdate = Boolean.valueOf(ReqDataIsDate[1][i]);
            int intIsDate = 0;
            if (isdate) {
                intIsDate = 1;
            }
            if (!isdate) {
                intIsDate = 0;
            }
            if (reqname.trim().equals("") == false) {
                String str2 = "INSERT INTO ReqDataIsDate VALUES (\'" +
                    reqname + "\',\'" + intIsDate + "\')";
                stmt3.executeUpdate(str2);
                System.out.println(str2);
            }
        }
        for (int i = 0; i < L6; i++) {
            String resname = ResDataIsDate[0][i];
            Boolean isdate = Boolean.valueOf(ResDataIsDate[1][i]);
            int intIsDate = 0;
            if (isdate) {
                intIsDate = 1;
            }
            if (!isdate) {
```

```
            intIsDate = 0;
        }
        if (resname.trim().equals("") == false) {
            String str2 = "INSERT INTO ResDataIsDate VALUES (\'" +
                resname + "\',\'" + intIsDate + "\')";
            stmt4.executeUpdate(str2);
            System.out.println(str2);
        }
    }
    for (int i = 0; i < L7; i++) {
        String reqname = ReqDataIsNum[0][i];
        Boolean isNum = Boolean.valueOf(ReqDataIsNum[1][i]);
        int intIsNum = 0;
        if (isNum) {
            intIsNum = 1;
        }
        if (!isNum) {
            intIsNum = 0;
        }
        if (reqname.trim().equals("") == false) {
            String str2 = "INSERT INTO ReqDataIsNum VALUES (\'" +
                reqname + "\',\'" + intIsNum + "\')";
            stmt3.executeUpdate(str2);
            System.out.println(str2);
        }
    }
    for (int i = 0; i < L8; i++) {
        String resname = ResDataIsNum[0][i];
        Boolean isNum = Boolean.valueOf(ResDataIsNum[1][i]);
        int intIsNum = 0;
        if (isNum) {
            intIsNum = 1;
        }
        if (!isNum) {
            intIsNum = 0;
        }
        if (resname.trim().equals("") == false) {
            String str2 = "INSERT INTO ResDataIsNum VALUES (\'" +
                resname + "\',\'" + intIsNum + "\')";
            stmt4.executeUpdate(str2);
            System.out.println(str2);
        }
    }
```

```
        for (int i = 0; i < L9; i++) {
            String reqname = ReqDataHasChar[0][i];
            Boolean isHasChar = Boolean.valueOf(ReqDataHasChar[1][i]);
            int intHasChar = 0;
            if (isHasChar) {
                intHasChar = 1;
            }
            if (!isHasChar) {
                intHasChar = 0;
            }
            if (reqname.trim().equals("") == false) {
                String str2 = "INSERT INTO ReqDataHasChar VALUES (\'" +
                    reqname + "\',\'" + intHasChar + "\')";
                stmt3.executeUpdate(str2);
                System.out.println(str2);
            }
        }
        for (int i = 0; i < L10; i++) {
            String resname = ResDataHasChar[0][i];
            Boolean isHasChar = Boolean.valueOf(ResDataHasChar[1][i]);
            int intHasChar = 0;
            if (isHasChar) {
                intHasChar = 1;
            }
            if (!isHasChar) {
                intHasChar = 0;
            }
            if (resname.trim().equals("") == false) {
                String str2 = "INSERT INTO ResDataHasChar VALUES (\'" +
                    resname + "\',\'" + intHasChar + "\')";
                stmt4.executeUpdate(str2);
                System.out.println(str2);
            }
        }
        try {
            String reqLenMin = String.valueOf(ReqSOAPLen[0]);
            String reqLenMax = String.valueOf(ReqSOAPLen[1]);
            String resLenMin = String.valueOf(ResSOAPLen[0]);
            String resLenMax = String.valueOf(ResSOAPLen[1]);
            stmt1.executeUpdate("INSERT INTO ReqSOAPLen VALUES (\'" +
                reqLenMin + "\',\'" + reqLenMax + "\')");
            stmt2.executeUpdate("INSERT INTO ResSOAPLen VALUES (\'" +
                resLenMin + "\',\'" + resLenMax + "\')");
```

```
        } catch (Exception e) {
            e.printStackTrace();
        }
        for (int i = 0; i < L11; i++) {

            try {
                String reqEncName = ReqEncodingList[i];
                if (reqEncName.trim().equals("") == false) {
                    String str1 = "INSERT INTO ReqEncodingList VALUES (\'"
                        + reqEncName + "\')";
                    stmt1.executeUpdate(str1);
                    System.out.println(str1);
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        try {
            for (int i = 0; i < L12; i++) {
                String resEncName = ResEncodingList[i];
                if (resEncName.trim().equals("") == false) {
                    String str1 = "INSERT INTO ResEncodingList VALUES (\'"
                        + resEncName + "\')";
                    stmt1.executeUpdate(str1);
                    System.out.println(str1);
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        for (int i = 0; i < L13; i++) {
            try {
                String reqname = RequestNameCountRange[0][i];
                String reqmin = RequestNameCountRange[1][i];
                String reqmax = RequestNameCountRange[2][i];
                if (reqname.trim().equals("") == false) {
                    String str1 = "INSERT INTO ReqNameCountRange VALUES (\'"
                        + reqname + "\',\'" + reqmin + "\',\'" +
                        reqmax + "\')";
                    stmt1.executeUpdate(str1);
                    System.out.println(str1);
                }
            } catch (Exception e) {
```

```
                e.printStackTrace();
            }
        }
        for (int i = 0; i < L14; i++) {
            try {
                String resname = ResponseNamesCountRange[0][i];
                String resmin = ResponseNamesCountRange[1][i];
                String resmax = ResponseNamesCountRange[2][i];
                if (resname.trim().equals("") == false) {
                    String str1 = "INSERT INTO ResNameCountRange VALUES ('"
                        + resname + "','" + resmin + "','" +
                        resmax + "')";
                    stmt1.executeUpdate(str1);
                    System.out.println(str1);
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

B.3 Learning an XML Tag Counts Range

This function learns the minimum and maximum possible number of occurrences of every XML tag in a SOAP message.

```
static String[][] LearnTagsCountRange(Connection con, String DBName,
    String VarTable, String VarNameCol, String VarValueCol,
    String IdColName) {
    //This routine learns the minimum and maximum counts of a all
    //xml tags in all soap messages
    // it first makes a list of all possible xml tags
    // then starts checking the count of each xml tag in
    //every single massage
    // then calculates the minimum occurancy of each xml
    //tag in all messages
    // then calculates the maximum occurancy of each xml
```

```
//tag in all messages
Statement stmt;
Statement stmt2;
String[][] retval;
try {
   stmt = con.createStatement();
   stmt.executeUpdate("use " + DBName);
   stmt2 = con.createStatement();
   stmt2.executeUpdate("use " + DBName);
   ResultSet AllIdsCol = stmt.executeQuery("SELECT " + IdColName
      + " FROM " + VarTable);
   ResultSet AllNamesCol = stmt2.executeQuery("SELECT " + VarNameCol
      + " FROM " + VarTable);
   java.util.ArrayList UniqueIds = FindUniqueNames(AllIdsCol);
   java.util.ArrayList UniqueNames = FindUniqueNames(AllNamesCol);
   java.lang.Object[] UniqueIdsO = UniqueIds.toArray();
   java.lang.Object[] AllUniqueNamesO = UniqueNames.toArray();
   int LAllUniqId = UniqueIdsO.length;
   int LAllUniqNames = AllUniqueNamesO.length;
   retval = new String[3][LAllUniqNames];
   //initialize the variable retval
   for (int j = 0; j < LAllUniqNames; j++) {
      retval[0][j] = AllUniqueNamesO[j].toString();
      retval[1][j] = Integer.toString(Integer.MAX_VALUE);
      retval[2][j] = Integer.toString(Integer.MIN_VALUE);
   }
   for (int i = 0; i < LAllUniqId; i++) {
      try {
         String Id = (String) UniqueIdsO[i];
         //select from the variables table database all varnames
         //(such as 'FROM') with the same message id
         //repeat this process for all possible xml tags to find
         //the minimum and maximum occurancies for each
         //xml tag (varname)
         String str = "SELECT " + VarNameCol + " FROM " + VarTable
            + " WHERE " + IdColName + " = \'" + Id + "\'";
         ResultSet rs2 = stmt.executeQuery(str);
         java.util.ArrayList UniqVarsNames = FindUniqueNames(rs2);
         java.lang.Object[] UniqVarsNamesO = UniqVarsNames.toArray();
         ResultSet rs3 = stmt.executeQuery(str);
         ArrayList al = ConvertRStoArrayList(rs3);
         java.lang.Object[] AllVarsNamesO = al.toArray();
         int LV = AllVarsNamesO.length;
```

```
                int LUV = UniqVarsNamesO.length;
                int[] VarCnt = ElementsCount(UniqVarsNamesO, AllVarsNamesO);
                try {
                   for (int j = 0; j < LAllUniqNames; j++) {
                      for (int k = 0; k < LUV; k++) {
                         if (UniqVarsNamesO[k].toString().equals(
                            AllUniqueNamesO[j].toString())) {
                            if (VarCnt[k] != 0) {
                               try {
                                  retval[1][j] = String.valueOf(
                                     java.lang.Math.min(Integer.valueOf(
                                     retval[1][j]), VarCnt[k]));
                                  retval[2][j] = String.valueOf(
                                     java.lang.Math.max(Integer.valueOf(
                                     retval[2][j]), VarCnt[k]));
                               } catch (Exception e) {
                               }
                            }
                         }
                      }
                   }
                } catch (Exception e) {
                   e.printStackTrace();
                }
             } catch (Exception e) {
                e.printStackTrace();
             }
          }
          return retval;
       } catch (Exception e) {
          e.printStackTrace();
       }
       return null;
    }
```

This is a supporting function used by the previous function that learns the minimum and maximum possible number of occurrences of each XML tag in a SOAP message.

```
static int[] ElementsCount(Object[] UniqNames, Object[] Data) {
   // counts the number of UniqNames[i] in Data array
   // the return value is the counts of UniqNames[i] in
   //Data[j] as array of
   // the same length as UniqNames array
   int L = UniqNames.length;
   int LD = Data.length;
   int[] retval = new int[L];
   java.util.Arrays.fill(retval, 0);
   for (int i = 0; i < L; i++) {
      for (int j = 0; j < LD; j++) {
         try {
            if (Data[j].toString().equals(UniqNames[i])) {
               retval[i] = retval[i] + 1;
            }
         } catch (Exception e) {
            e.printStackTrace();
         }
      }
   }
   return retval;
}
```

B.4 Learning Calls Dependencies

This function learns the list of XML tags requests that must/may precede every XML response tag.

```
static Object[] LearnCallsSequence(Connection con, String DBName,
   String ReqValsTable, String ResValsTable, String ReqValNameCol,
   String ReqValValueCol, String ResValNameCol, String ResValValueCol) {
   Object[] O = new Object[6];
   try {
      Statement stmt1;
      Statement stmt2;
      stmt1 = con.createStatement();
      stmt2 = con.createStatement();
```

```
stmt1.executeUpdate("use " + DBName);
stmt2.executeUpdate("use " + DBName);
//Read RequestId and ResponseId list for all requests
//and responses
//Read xml tags (Request and response variables names)
//for all reqs and resps
//We get four ResultSet's
ResultSet ResponseIdRS = stmt1.executeQuery("SELECT
   ResponseId FROM SoapIDSResponseTable");
ResultSet RequestIdRS = stmt2.executeQuery("SELECT
   RequestId FROM SoapIDSRequestTable");
ResultSet ResponseVarNameRS = stmt1.executeQuery("SELECT
   VarName FROM SoapIDSResponseVarsTable");
ResultSet RequestVarNameRS = stmt2.executeQuery("SELECT
   VarName FROM SoapIDSRequestVarsTable");
//Make a list of unique IDs and names for the previous queries
//We get four ArrayList's
ArrayList ResponseIdArr = FindUniqueNames(ResponseIdRS);
ArrayList RequestIdArr = FindUniqueNames(RequestIdRS);
ArrayList ResponseVarNameArr = FindUniqueNames(ResponseVarNameRS);
ArrayList RequestVarNameArr = FindUniqueNames(RequestVarNameRS);
//Remove empty array members generated because of lousy coding
Object[] ResponseIdArr2 =
   RemoveEmptyMembers(ResponseIdArr).toArray();
Object[] RequestIdArr2 =
   RemoveEmptyMembers(RequestIdArr).toArray();
Object[] ResponseVarNameArr2 =
   RemoveEmptyMembers(ResponseVarNameArr).toArray();
Object[] RequestVarNameArr2 =
   RemoveEmptyMembers(RequestVarNameArr).toArray();
//Sort the unique read data alphabatically
//Unique: RequestId, ResponseId, RequestVarName, ResponseVarName
Arrays.sort(ResponseIdArr2);
Arrays.sort(RequestIdArr2);
Arrays.sort(ResponseVarNameArr2);
Arrays.sort(RequestVarNameArr2);
//Get the length of each array of the four arrays after
//removing duplicates and empty members
int LenResponseId = ResponseIdArr2.length;
int LenRequestId = RequestIdArr2.length;
int LenResponseVarName = ResponseVarNameArr2.length;
int LenRequestVarName = RequestVarNameArr2.length;
//Make a hashmap for every array
```

```
java.util.HashMap ResponseIdMap = new java.util.HashMap();
// Map of all ResponseId is saved here
java.util.HashMap RequestIdMap = new java.util.HashMap();
// Map of all RequestId is saved here
java.util.HashMap ResponseVarNameMap = new java.util.HashMap();
// Map of all response var names
java.util.HashMap RequestVarNameMap = new java.util.HashMap();
// Map of all request var names
//Save in each hashmap two values
//For the ResponseId hashmap save the responseid itself in
//the first column and an index starting from zero in
//the 2nd column
//Do the same for the requestid hashmap and the
//RequestVarName and ResponseVarName hashmaps
for (int i = 0; i < LenResponseId; i++) {
   ResponseIdMap.put(ResponseIdArr2[i], i);
}
for (int i = 0; i < LenRequestId; i++) {
   RequestIdMap.put(RequestIdArr2[i], i);
}
for (int i = 0; i < LenResponseVarName; i++) {
   ResponseVarNameMap.put(ResponseVarNameArr2[i], i);
}
for (int i = 0; i < LenRequestVarName; i++) {
   RequestVarNameMap.put(RequestVarNameArr2[i], i);
}
// Request and Response var names and their IDs are
//now saved and ready
//DependenciesTable is a table that will later
//contain the relationship for every single soap
//request/response pair
boolean DependenciesTable[][][] =
   new boolean
   [LenRequestVarName][LenResponseVarName][LenRequestId];
//NANDTable and NORTable will contain a  summary for
//the data contained in DependenciesTable
//NANDTable is
//NORTable is
//boolean ANDTable[][] =
//new boolean[LenRequestVarName][LenResponseVarName];
//boolean ORTable[][] =
//new boolean[LenRequestVarName][LenResponseVarName];
//NORTable and NANDTable are not used but
```

```
//left here to avoid any compilation error
//lousy coding again in the works here!!
boolean NORTable[][] =
   new boolean[LenRequestVarName][LenResponseVarName];
boolean NANDTable[][] =
   new boolean[LenRequestVarName][LenResponseVarName];
String ReqNamesList[] =
   new String[LenRequestVarName];
String ResNamesList[] =
   new String[LenResponseVarName];
//start a loop over all request id's
//The goal is to fill DependenciesTable after looping
//over all of the collected request and responses data
for (int n = 0; n < LenRequestId; n++) {
   try {
      String RequestIDStr = RequestIdArr2[n].toString();
      //Read from the database all request/response
      //varnames with a given request id only
      ResultSet ResVarNameRS = stmt2.executeQuery("SELECT
         VarName FROM SoapIDSResponseVarsTable WHERE
         ResponseId = \'" + RequestIDStr + "\'");
      ResultSet ReqVarNameRS = stmt1.executeQuery("SELECT
         VarName FROM SoapIDSRequestVarsTable WHERE
         RequestId = \'" + RequestIDStr + "\'");
      ArrayList ReqVarNameArr = FindUniqueNames(ReqVarNameRS);
      ArrayList ResVarNameArr = FindUniqueNames(ResVarNameRS);
      java.lang.Object[] CurrentReqVarNameArr =
         RemoveEmptyMembers(ReqVarNameArr).toArray();
      java.lang.Object[] CurrentResVarNameArr =
         RemoveEmptyMembers(ResVarNameArr).toArray();
      Arrays.sort(CurrentReqVarNameArr);
      Arrays.sort(CurrentResVarNameArr);
      int LenCurrentReqVarNameArr = CurrentReqVarNameArr.length;
      int LenCurrentResVarNameArr = CurrentResVarNameArr.length;
      int r = Integer.valueOf(
         RequestIdMap.get(RequestIDStr).toString());
         //r: index of request Id
      for (int i = 0; i < LenCurrentReqVarNameArr; i++) {
         int p = Integer.valueOf(RequestVarNameMap.get(
            CurrentReqVarNameArr[i]).toString());
            //p: index of request var name
         ReqNamesList[p] = CurrentReqVarNameArr[i].toString();
         for (int j = 0; j < LenCurrentResVarNameArr; j++) {
```

```
                int q = Integer.valueOf(ResponseVarNameMap.get(
                    CurrentResVarNameArr[j]).toString());
                    //q: index of response var name
                ResNamesList[q] = CurrentResVarNameArr[j].toString();
                DependenciesTable[p][q][r] = true;
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
boolean res_exist_in_id[][] =
    new boolean[LenResponseVarName][LenRequestId];
for (int j = 0; j < LenResponseVarName; j++) {
    for (int k = 0; k < LenRequestId; k++) {
        res_exist_in_id[j][k] =
            Response_exist_in_id(j, k, DependenciesTable);
    }
}
boolean req_exist_in_id[][] =
    new boolean[LenRequestVarName][LenRequestId];
for (int k = 0; k < LenRequestId; k++) {
    for (int i = 0; i < LenRequestVarName; i++) {
        req_exist_in_id[i][k] =
            Request_exist_in_id(i, k, DependenciesTable);
    }
}
boolean res_always_preceded_by_req[][] =
    new boolean[LenRequestVarName][LenResponseVarName];
boolean res_may_preceded_by_req[][] =
    new boolean[LenRequestVarName][LenResponseVarName];
    new boolean[LenRequestVarName][LenResponseVarName];
for (int ResIdx = 0; ResIdx < LenResponseVarName; ResIdx++) {
    for (int ReqIdx = 0; ReqIdx < LenRequestVarName; ReqIdx++) {
        res_always_preceded_by_req[ReqIdx][ResIdx] =
            is_res_always_preceded_by_req(ResIdx, ReqIdx,
            req_exist_in_id, res_exist_in_id, LenResponseVarName,
            LenRequestVarName, LenRequestId);
        res_may_preceded_by_req[ReqIdx][ResIdx] =
            is_res_may_preceded_by_req(ResIdx, ReqIdx,
            req_exist_in_id, res_exist_in_id, LenResponseVarName,
            LenRequestVarName, LenRequestId);
    }
```

```
            }
            O[0] = res_always_preceded_by_req;
            O[1] = res_may_preceded_by_req;
            O[2] = NORTable;
            O[3] = NANDTable;
            O[4] = ReqNamesList;
            O[5] = ResNamesList;
            return O;
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }
```

This is a supporting function used by the previous function that learns the request/response dependencies.

```
static boolean is_res_always_preceded_by_req(int ResIdx,
   int ReqIdx, boolean req_exist_in_id[][], boolean res_exist_in_id[][],
      int LenResponseVarName, int LenRequestVarName, int LenRequestId) {
   boolean always_preceded = true;
   try {
      for (int id = 0; id < LenRequestId; id++) {
         if (res_exist_in_id[ResIdx][id] == true) {
            always_preceded = always_preceded &&
               req_exist_in_id[ReqIdx][id];
         }
      }
   } catch (Exception e) {
      e.printStackTrace();
   }
   return always_preceded;
}
```

This is a supporting function used by the previous function that learns the request/response dependencies.

```
static boolean is_res_may_preceded_by_req(int ResIdx, int ReqIdx,
   boolean req_exist_in_id[][], boolean res_exist_in_id[][],
      int LenResponseVarName, int LenRequestVarName, int LenRequestId) {
   boolean may_preceded = false;
   try {
      for (int id = 0; id < LenRequestId; id++) {
         if (res_exist_in_id[ResIdx][id] == true) {
            may_preceded = may_preceded || req_exist_in_id[ReqIdx][id];
         }
      }
   } catch (Exception e) {
      e.printStackTrace();
   }
   return may_preceded;
}
```

This is a supporting function used by the previous function that learns the request/response dependencies.

```
static boolean Request_exist_in_id(int ReqIdx, int Id,
   boolean DependenciesTable[][][]) {
   boolean exist = DependenciesTable[ReqIdx][0][Id];
   int Li = DependenciesTable.length;
   int Lj = DependenciesTable[0].length;
   for (int j = 0; j < Lj; j++) {
      exist = exist || DependenciesTable[ReqIdx][j][Id];
   }
   return exist;
}
```

This is a supporting function used by the previous function that learns the request/response dependencies.

```
static boolean Response_exist_in_id(int ResIdx, int Id,
      boolean DependenciesTable[][][]) {
   boolean exist = DependenciesTable[0][ResIdx][Id];
   int Li = DependenciesTable.length;
   int Lj = DependenciesTable[0].length;
   for (int i = 0; i < Li; i++) {
      exist = exist || DependenciesTable[i][ResIdx][Id];
   }
   return exist;
}
```

This is a supporting function used by the previous function that learns the request/response dependencies.

```
static ArrayList RemoveEmptyMembers(ArrayList inarr) {
   java.util.Collection c = new java.util.HashSet();
   c.add("");
   c.add(" ");
   c.add("  ");
   c.add("   ");
   inarr.removeAll(c);
   return inarr;
}
```

B.5 Learning Messages Encodings

This function learns the encoding of the request/response SOAP messages. This function is not complete yet. It is listed here as a reminder of more future work to be done here.

```
static String[] LearnEncoding(Connection con, String DBName,
   String Table, String EncodingCol) {
   Statement stmt;
   try {
      stmt = con.createStatement();
      stmt.executeUpdate("use " + DBName);
      ResultSet EncCol = stmt.executeQuery("SELECT " +
         EncodingCol + " FROM " + Table);
      ArrayList arr = FindUniqueNames(EncCol);
      Object[] arr2 = arr.toArray();
      String[] retval;
      int L = arr2.length;
      retval = new String[L];
      for (int i = 0; i < L; i++) {
         retval[i] = (String) arr2[i];
      }
      return retval;
   } catch (Exception e) {
      e.printStackTrace();
   }
   return null;
}
```

B.6 Learning Messages Lengths

This function learns the minimum and maximum possible length of all SOAP messages.

```
static int[] LearnSOAPMessageLength(Connection con,
   String DBName, String ReqTable, String LengthCol) {
   int[] retval;
   Statement stmt;
   try {
      stmt = con.createStatement();
      stmt.executeUpdate("use " + DBName);
      ResultSet AllVarNameCol = stmt.executeQuery(
         "SELECT " + LengthCol + " FROM " + ReqTable);
      retval = GetArrayMinMax(AllVarNameCol);
```

```
            return retval;
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }
```

B.7 Learning Allowed Special Characters Set

This functions learns whether a given set of characters appear in any given SOAP message.

```
static String[][] LearnHasChar(Connection con, String DBName,
    String VarTable, String VarNameCol, String VarValueCol,
    String CharSet) {
    Statement stmt;
    String[][] retval;
    try {
        stmt = con.createStatement();
        stmt.executeUpdate("use " + DBName);
        ResultSet AllVarNameCol = stmt.executeQuery("SELECT "
            + VarNameCol + " FROM " + VarTable);
        java.util.ArrayList UniqueVarsNames =
            FindUniqueNames(AllVarNameCol);
        java.lang.Object[] UniqueVarsNamesO =
            UniqueVarsNames.toArray();
        int L = UniqueVarsNamesO.length;
        retval = new String[2][L];
        for (int i = 0; i < L; i++) {
            try {
                String VarName = (String) UniqueVarsNamesO[i];
                retval[0][i] = VarName;
                retval[1][i] = String.valueOf(false);
                String str = "SELECT " + VarValueCol + " FROM " +
                    VarTable + " WHERE " + VarNameCol + " = \'" +
                    VarName + "\'";
                ResultSet rs2 = stmt.executeQuery(str);
                java.util.ArrayList VarsValues = FindUniqueNames(rs2);
                java.lang.Object[] VarsValuesO = VarsValues.toArray();
                int LV = VarsValuesO.length;
                boolean CanHasCharSet = false;
                for (int j = 0; j < LV; j++) {
                    try {
```

```
                    CanHasCharSet = CanHasCharSet || ContainsCharSet(
                        VarsValuesO[j].toString(), CharSet);
                    retval[1][i] = String.valueOf(CanHasCharSet);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
            System.out.println(retval[0][i] + "\t" + retval[1][i]);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    return retval;
} catch (Exception e) {
    e.printStackTrace();
}
return null;
}
```

B.8 Learning if XML Tag Value can be Casted to a Number

This function learns whether an XML tag can always be casted to a number or not.

```
static String[][] LearnCastNum(Connection con, String DBName,
    String VarTable, String VarNameCol, String VarValueCol) {
    Statement stmt;
    String[][] retval;
    try {
        stmt = con.createStatement();
        stmt.executeUpdate("use " + DBName);
        ResultSet AllVarNameCol = stmt.executeQuery("SELECT " +
            VarNameCol + " FROM " + VarTable);
        java.util.ArrayList UniqueVarsNames =
            FindUniqueNames(AllVarNameCol);
        java.lang.Object[] UniqueVarsNamesO = UniqueVarsNames.toArray();
        int L = UniqueVarsNamesO.length;
        retval = new String[2][L];
        for (int i = 0; i < L; i++) {
            try {
                String VarName = (String) UniqueVarsNamesO[i];
                retval[0][i] = VarName;
                retval[1][i] = String.valueOf(false);
```

```
                String str = "SELECT " + VarValueCol + " FROM " +
                    VarTable + " WHERE " + VarNameCol + " = \'" +
                    VarName + "\'";
                ResultSet rs2 = stmt.executeQuery(str);
                java.util.ArrayList VarsValues = FindUniqueNames(rs2);
                java.lang.Object[] VarsValuesO = VarsValues.toArray();
                int LV = VarsValuesO.length;
                boolean IsNumSoFar = true;
                for (int j = 0; j < LV; j++) {
                    try {
                        IsNumSoFar = IsNumSoFar &&
                            IsNum(VarsValuesO[j].toString());
                        retval[1][i] = String.valueOf(IsNumSoFar);
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
                System.out.println(retval[0][i] + "\t" + retval[1][i]);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        return retval;
    } catch (Exception e) {
        e.printStackTrace();
    }
    return null;
}
```

B.9 Learning if XML Tag Value can be Casted to a Date/Time

This function learns whether an XML tag can always be casted to a Date/Time or not. This function still needs more work. It is listed here as a reminder of more future work that is needed to be done here.

```
static String[][] LearnCastDate(Connection con, String DBName,
    String VarTable, String VarNameCol, String VarValueCol) {
    Statement stmt;
    String[][] retval;
    try {
        stmt = con.createStatement();
        stmt.executeUpdate("use " + DBName);
        ResultSet AllVarNameCol = stmt.executeQuery("SELECT " +
```

```
            VarNameCol + " FROM " + VarTable);
        java.util.ArrayList UniqueVarsNames =
            FindUniqueNames(AllVarNameCol);
        java.lang.Object[] UniqueVarsNamesO =
            UniqueVarsNames.toArray();
        int L = UniqueVarsNamesO.length;
        retval = new String[2][L];
        for (int i = 0; i < L; i++) {
            try {
                String VarName = (String) UniqueVarsNamesO[i];
                retval[0][i] = VarName;
                retval[1][i] = String.valueOf(false);
                String str = "SELECT " + VarValueCol + " FROM " + VarTable +
                    " WHERE " + VarNameCol + " = \'" + VarName + "\'";
                ResultSet rs2 = stmt.executeQuery(str);
                java.util.ArrayList VarsValues = FindUniqueNames(rs2);
                java.lang.Object[] VarsValuesO = VarsValues.toArray();
                int LV = VarsValuesO.length;
                boolean IsDateSoFar = true;
                for (int j = 0; j < LV; j++) {
                    try {
                        IsDateSoFar = IsDateSoFar &&
                            IsDate(VarsValuesO[j].toString());
                        retval[1][i] = String.valueOf(IsDateSoFar);
                        //System.out.println(VarName + "\t" + VarsValuesO[j].
                        //toString() + "\t" + String.valueOf(Len));
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
                System.out.println(retval[0][i] + "\t" + retval[1][i]);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        return retval;
    } catch (Exception e) {
        e.printStackTrace();
    }
    return null;
}
```

B.10 Learning if XML Tag Value can be Casted to a Boolean

This function learns whether an XML tag can always be casted to a Boolean or not.

```
static String[][] LearnCastBool(Connection con, String DBName,
    String VarTable, String VarNameCol, String VarValueCol) {
    Statement stmt;
    String[][] retval;
    try {
        stmt = con.createStatement();
        stmt.executeUpdate("use " + DBName);
        ResultSet AllVarNameCol = stmt.executeQuery("SELECT " +
            VarNameCol + " FROM " + VarTable);
        java.util.ArrayList UniqueVarsNames =
            FindUniqueNames(AllVarNameCol);
        java.lang.Object[] UniqueVarsNamesO =
            UniqueVarsNames.toArray();
        int L = UniqueVarsNamesO.length;
        retval = new String[2][L];
        for (int i = 0; i < L; i++) {
            try {
                String VarName = (String) UniqueVarsNamesO[i];
                retval[0][i] = VarName;
                retval[1][i] = String.valueOf(false);
                String str = "SELECT " + VarValueCol + " FROM " + VarTable +
                    " WHERE " + VarNameCol + " = \'" + VarName + "\'";
                ResultSet rs2 = stmt.executeQuery(str);
                java.util.ArrayList VarsValues = FindUniqueNames(rs2);
                java.lang.Object[] VarsValuesO = VarsValues.toArray();
                int LV = VarsValuesO.length;
                boolean IsBoolSoFar = true;
                for (int j = 0; j < LV; j++) {
                    try {
                        IsBoolSoFar = IsBoolSoFar &&
                            IsBoolean(VarsValuesO[j].toString());
                        retval[1][i] = String.valueOf(IsBoolSoFar);
                        //System.out.println(VarName + "\t" +
                        //VarsValuesO[j].toString() + "\t" +
                        //String.valueOf(Len));
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
```

```
                System.out.println(retval[0][i] + "\t" + retval[1][i]);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        return retval;
    } catch (Exception e) {
        e.printStackTrace();
    }
    return null;
}
```

B.11 Learning XML Tags Values Lengths

This function learns the minimum and maximum possible lengths of each XML tag value in all SOAP messages.

```
static String[][] LearnDataLength(Connection con, String DBName,
    String VarTable, String VarNameCol, String VarValueCol) {
    Statement stmt;
    String[][] retval;
    try {
        stmt = con.createStatement();
        stmt.executeUpdate("use " + DBName);
        ResultSet AllVarNameCol = stmt.executeQuery("SELECT " +
            VarNameCol + " FROM " + VarTable);
        java.util.ArrayList UniqueVarsNames=FindUniqueNames(AllVarNameCol);
        java.lang.Object[] UniqueVarsNamesO=UniqueVarsNames.toArray();
        int L = UniqueVarsNamesO.length;
        retval = new String[3][L];
        for (int i = 0; i < L; i++) {
            try {
                retval[1][i] = String.valueOf(Integer.MAX_VALUE);
                retval[2][i] = String.valueOf(Integer.MIN_VALUE);
                String VarName = (String) UniqueVarsNamesO[i];
                String str = "SELECT " + VarValueCol + " FROM " +
                VarTable +" WHERE "+VarNameCol + " = \'" + VarName+"\'";
                ResultSet rs2 = stmt.executeQuery(str);
                java.util.ArrayList VarsValues = FindUniqueNames(rs2);
                java.lang.Object[] VarsValuesO = VarsValues.toArray();
                int LV = VarsValuesO.length;
                retval[0][i] = VarName;
```

```
            for (int j = 0; j < LV; j++) {
               try {
                  int Len = VarsValuesO[j].toString().length();
                  int minlen = Integer.valueOf(retval[1][i]);
                  int maxlen = Integer.valueOf(retval[2][i]);
                  retval[1][i] = String.valueOf(java.lang.Math.min(
                     minlen, Len));
                  retval[2][i] = String.valueOf(java.lang.Math.max(
                     maxlen, Len));
                  //System.out.println(VarName + "\t" +
                  //VarsValuesO[j].toString()+"\t"+String.valueOf(Len));
               } catch (Exception e) {
                  e.printStackTrace();
               }
            }
            System.out.println(retval[0][i] + "\t" + retval[1][i] +
               "\t" + retval[2][i]);
         } catch (Exception e) {
            e.printStackTrace();
         }
      }
      return retval;
   } catch (Exception e) {
      e.printStackTrace();
   }
   return null;
}
```

B.12 More Supporting Common Functions

This section lists the functions that are common to all of the previous functions.

```
static int[] GetArrayMinMax(ResultSet rs) {
   //find the smallest and largest number in a column
   //of type ResultSet that contains only numbers
   //returns an in array of length 2
   //int[0] is the minimum
   //int[1] is the maximum
   java.util.ArrayList arr = ConvertResultSetToIntArray(rs);
   int[] retval = new int[2];
   int minval = Integer.MAX_VALUE;
```

```
    int maxval = Integer.MIN_VALUE;
    try {
       Object[] arr2 = arr.toArray();
       int L = arr2.length;
       for (int i = 0; i < L; i++) {
          try {
             minval = java.lang.Math.min(minval,
                Integer.parseInt(arr2[i].toString()));
             maxval = java.lang.Math.max(maxval,
                Integer.parseInt(arr2[i].toString()));
          } catch (Exception e) {
             e.printStackTrace();
          }
       }
    } catch (Exception e) {
       e.printStackTrace();
    }
    retval[0] = minval;
    retval[1] = maxval;
    return retval;
 }
```

```
 static java.util.ArrayList ConvertResultSetToIntArray(ResultSet rs) {
    //Convert a column of ResultSet that contains only numbers to
    //a column of integers
    java.util.ArrayList<Integer> OutArray =
       new java.util.ArrayList<Integer>();
    try {
       while (rs.next()) {
          int val = rs.getInt(1);
          OutArray.add(val);
       }
    } catch (Exception e) {
       e.printStackTrace();
    }
    return OutArray;
 }
```

```
static java.util.ArrayList ConvertRStoArrayList(ResultSet InRS) {
   //Convert ResultSet to ArrayList
   //ResultSet is the return value of SQL queries in java
   if (InRS == null) {
      return null;
   }
   java.util.ArrayList<String> OutArray =
      new java.util.ArrayList<String>();
   try {
      while (InRS.next()) {
         String val = InRS.getString(1);
         OutArray.add(val);
      }
   } catch (SQLException e) {
      e.printStackTrace();
   }
   return OutArray;
}
```

```
static java.util.ArrayList FindUniqueNames(ResultSet InRS) {
   //find all unique names in the column InRS of type ResultSet
   // and return the result as ArrayList
   if (InRS == null) {
      return null;
   }
   java.util.ArrayList<String> OutArray =
      new java.util.ArrayList<String>();
   try {
      while (InRS.next()) {
         String val = InRS.getString(1);
         if (!OutArray.contains(val)) {
            OutArray.add(val);
         }
      }
   } catch (SQLException e) {
      e.printStackTrace();
   }
   return OutArray;
}
```

```
static boolean IsNum(String s) {
   //Is the string s a number?
   try {
      double d = Double.valueOf(s);
      return true;
   } catch (Exception e) {
      return false;
   }

}
```

```
static boolean IsTrue(String s) {
   //is string s boolean=true
   if (s.equalsIgnoreCase("true") || s.equalsIgnoreCase("1") ||
      s.equalsIgnoreCase("t") || s.equalsIgnoreCase("yes") ||
      s.equalsIgnoreCase("y")) {
      return true;
   }
   return false;
}
```

```
static boolean IsFalse(String s) {
   //is string s boolean=false
   if (s.equalsIgnoreCase("false") || s.equalsIgnoreCase("0") ||
      s.equalsIgnoreCase("f") || s.equalsIgnoreCase("no") ||
      s.equalsIgnoreCase("n")) {
      return true;
   }
   return false;
}
```

```
static boolean IsBoolean(String s) {
   //is string s boolean (can cast to boolean?)
   if (IsTrue(s) || IsFalse(s)) {
      return true;
   }
   return false;
}
```

```
    static boolean IsDate(String s) {
        //can i cast string s to Date?
        DateFormat df = new SimpleDateFormat();
        try {
            String s2 = s.trim();
            if (s2.isEmpty()) {
                return false;
            }
            java.util.Date d = df.parse(s2.trim());
            System.out.println(d);
            return true;
        } catch (Exception e) {
            return false;
        }
    }
```

```
    static boolean ContainsCharSet(String s1, String s2) {
        //check if string s1 contains s2
        try {
            return s1.contains(s2);
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }
}
```

APPENDIX C. Detection Phase Source Code

The tool that captures any possible intrusion is called wsmonitor-detect. The source code for this tool is the same as the source code for wsmonitor-collect tool (see appendix A) with few more functions that check the collected traffic against the learned characteristics. The code listed here is the code that is not part of wsmonitor-collect.

C.1 Checking Request Characteristics

This function is responsible for initiating the process of checking the captured request against the learned characteristics.

```
public static void InvestigateRequest(ParametersList params,
   Statement stmt, String Id,
      int RequestLength, String RequestEncoding, boolean
      RequestHasAttachment, ServiceCharacteristics Characteristics,
      IDSLogger TreeLogger) {
   IDSReport report = CheckAgainstRequestCharacteristics(params,
      RequestLength, RequestEncoding, RequestHasAttachment,
      Characteristics, TreeLogger);
}
```

```
private static IDSReport CheckAgainstRequestCharacteristics(
   ParametersList params, int RequestLength, String RequestEncoding,
   boolean RequestHasAttachment,
   ServiceCharacteristics Characteristics,
   IDSLogger iDSLogger) {
   IDSReport report = new IDSReport();
   if (RequestLength < Characteristics.ReqSOAPMin) {
      IDSEvent event = new IDSEvent();
      event.EventDate = new Date();
      event.EventLevel = 1;
      event.EventInfo = "A request SOAP message length (" +
         RequestLength + ") is less than the minimum learned length
```

```
            (" + Characteristics.ReqSOAPMin + ")";
        iDSLogger.ReportEvent(event);
    }
    if (RequestLength > Characteristics.ReqSOAPMax) {
        IDSEvent event = new IDSEvent();
        event.EventDate = new Date();
        event.EventLevel = 1;
        event.EventInfo = "A request SOAP message length (" +
            RequestLength + ") exceeds the maximum learned length ("
            + Characteristics.ReqSOAPMax + ")";
        iDSLogger.ReportEvent(event);
    }
    if (Characteristics.CheckIfReqEncodingIs(RequestEncoding) == false) {
        IDSEvent event = new IDSEvent();
        event.EventDate = new Date();
        event.EventLevel = 1;
        event.EventInfo = "A request SOAP message encoding (" +
            RequestEncoding + ") is different from learned encoding";
        iDSLogger.ReportEvent(event);
    }
    int ParamsCount = params.ParametersListName.size();
    for (int i = 0; i < ParamsCount; i++) {
        boolean b = false;
        b = Characteristics.CheckReqParameterNameExist(
            params.ParametersListName.get(i));
        if (b == false) {
            IDSEvent event = new IDSEvent();
            event.EventDate = new Date();
            event.EventLevel = 3;
            event.EventInfo = "A request SOAP tag (" +
                params.ParametersListName.get(i) + ") is not recognized";
            iDSLogger.ReportEvent(event);
        }
        b = Characteristics.CheckReqDataLength(
            params.ParametersListName.get(i),
            params.ParametersListValue.get(i));
        if (b == false) {
            IDSEvent event = new IDSEvent();
            event.EventDate = new Date();
            event.EventLevel = 2;
            inner:
            for (int n = 0;
                n < Characteristics.ReqDataLength[0].length; n++) {
```

```
            if (Characteristics.ReqDataLength[0][n].
               equalsIgnoreCase(params.ParametersListName.get(i))) {
               int min = Integer.valueOf(
                  Characteristics.ReqDataLength[1][n]);
               int max = Integer.valueOf(
                  Characteristics.ReqDataLength[2][n]);
               event.EventInfo = "A request SOAP tag value length("
                  + params.ParametersListValue.get(i).length() + ")
                  is not within the learned range (" + min + "," +
                  max + ")";
               iDSLogger.ReportEvent(event);
               break inner;
            }}}
      b = Characteristics.CheckReqDataIsBool(
         params.ParametersListValue.get(i));
      if (b == false) {
         IDSEvent event = new IDSEvent();
         event.EventDate = new Date();
         event.EventLevel = 1;
         event.EventInfo = "A request SOAP tag (" +
            params.ParametersListName.get(i) + ") value (" +
            params.ParametersListValue.get(i) + ") is not boolean
            as learned";
         iDSLogger.ReportEvent(event);
      }
      b = Characteristics.CheckReqDataIsDate(
         params.ParametersListValue.get(i));
      if (b == false) {
         IDSEvent event = new IDSEvent();
         event.EventDate = new Date();
         event.EventLevel = 1;
         event.EventInfo = "A request SOAP tag (" +
            params.ParametersListName.get(i) + ") value (" +
            params.ParametersListValue.get(i) + ") is not DATE-TIME
            as learned";
         iDSLogger.ReportEvent(event);
      }
      b = Characteristics.CheckReqDataIsNum(
         params.ParametersListValue.get(i));
      if (b == false) {
         IDSEvent event = new IDSEvent();
         event.EventDate = new Date();
         event.EventLevel = 1;
```

```
            event.EventInfo = "A request SOAP tag (" +
                params.ParametersListName.get(i) + ") value (" +
                params.ParametersListValue.get(i) + ") is expected to
                be numerical";
            iDSLogger.ReportEvent(event);
        }
        b = Characteristics.CheckReqDataHasChar(
            params.ParametersListValue.get(i));
        if (b == false) {
            IDSEvent event = new IDSEvent();
            event.EventDate = new Date();
            event.EventLevel = 1;
            event.EventInfo = "A request SOAP tag (" +
                params.ParametersListName.get(i) + ") value (" +
                params.ParametersListValue.get(i) + ") contained
                unexpected characters";
            iDSLogger.ReportEvent(event);
        }
        b = Characteristics.CheckReqNamesCountRange(
            params.ParametersListName.get(i), params);
        if (b == false) {
            IDSEvent event = new IDSEvent();
            event.EventDate = new Date();
            event.EventLevel = 3;
            inner:
            for (int n = 0; n < Characteristics.
                RequestNamesCountRange[0].length; n++) {
                if (Characteristics.RequestNamesCountRange[0][n].
                    equalsIgnoreCase(params.ParametersListName.get(i))) {
                    int min = Integer.
                    valueOf(Characteristics.RequestNamesCountRange[1][n]);
                    int max = Integer.valueOf(
                        Characteristics.RequestNamesCountRange[2][n]);
                    event.EventInfo = "A request SOAP tag frequency (" +
                        params.ParametersListName.get(i) + ") is not
                        within the learned range ("+ min +","+max + ")";
                    iDSLogger.ReportEvent(event);
                    break inner;
                }}}}
    return report;
}
```

C.2 Checking Response Characteristics

This function is responsible for initiating the process of checking the captured response against the learned characteristics.

```
public static void InvestigateResponse(ParametersList params,
      Statement stmt, String Id,
      int ResponseLength, String ResponseEncoding, boolean
      ResponseHasAttachment, ServiceCharacteristics Characteristics,
      IDSLogger TreeLogger) {
   IDSReport report = CheckAgainstResponseCharacteristics(params,
      ResponseLength, ResponseEncoding, ResponseHasAttachment,
      Characteristics, TreeLogger);
}
```

```
private static IDSReport CheckAgainstResponseCharacteristics(
   ParametersList resparams, int ResponseLength, String
   ResponseEncoding, boolean ResponseHasAttachment,
   ServiceCharacteristics Characteristics, IDSLogger iDSLogger) {
   IDSReport report = new IDSReport();
   if (ResponseLength < Characteristics.ResSOAPMin) {
      IDSEvent event = new IDSEvent();
      event.EventDate = new Date();
      event.EventLevel = 2;
      event.EventInfo = "A response SOAP message length (" +
         ResponseLength + ") is less than the minimum learned
         length (" + Characteristics.ResSOAPMin + ")";
      iDSLogger.ReportEvent(event);
   }
   if (ResponseLength > Characteristics.ResSOAPMax) {
      IDSEvent event = new IDSEvent();
      event.EventDate = new Date();
      event.EventLevel = 2;
      event.EventInfo = "A response SOAP message length (" +
         ResponseLength + ") exceeds the maximum learned length (" +
         Characteristics.ResSOAPMax + ")";
      iDSLogger.ReportEvent(event);
   }
   if (Characteristics.CheckIfResEncodingIs(ResponseEncoding)
      == false) {
      IDSEvent event = new IDSEvent();
      event.EventDate = new Date();
      event.EventLevel = 2;
```

```
        event.EventInfo = "A response SOAP message encoding (" +
            ResponseEncoding + ") is different from learned encoding";
        iDSLogger.ReportEvent(event);
    }
    int ParamsCount = resparams.ParametersListName.size();
    for (int i = 0; i < ParamsCount; i++) {
        boolean b = false;
        b = Characteristics.CheckResParameterNameExist(
            resparams.ParametersListName.get(i));
        if (b == false) {
            IDSEvent event = new IDSEvent();
            event.EventDate = new Date();
            event.EventLevel = 4;
            event.EventInfo = "A response SOAP tag (" +
                resparams.ParametersListName.get(i) + ") is
                not recognized";
            iDSLogger.ReportEvent(event);
        }
        b = Characteristics.CheckResDataLength(
            resparams.ParametersListName.get(i),
            resparams.ParametersListValue.get(i));
        if (b == false) {
            IDSEvent event = new IDSEvent();
            event.EventDate = new Date();
            event.EventLevel = 3;
            inner:
            for (int n = 0; n <
                Characteristics.ResDataLength[0].length; n++) {
                if (Characteristics.ResDataLength[0][n].equalsIgnoreCase(
                    resparams.ParametersListName.get(i))) {
                    int min = Integer.valueOf(
                        Characteristics.ResDataLength[1][n]);
                    int max = Integer.valueOf(
                        Characteristics.ResDataLength[2][n]);
                    event.EventInfo = "A response SOAP tag value length("
                        + resparams.ParametersListName.get(i).length()
                        + ") is not within the learned range (" + min +
                        "," + max + ")";
                    iDSLogger.ReportEvent(event);
                    break inner;
                }
            }
        }
```

```
b = Characteristics.CheckResDataIsBool(
    resparams.ParametersListValue.get(i));
if (b == false) {
    IDSEvent event = new IDSEvent();
    event.EventDate = new Date();
    event.EventLevel = 2;
    event.EventInfo = "A response SOAP tag (" +
        resparams.ParametersListName.get(i) + ") value (" +
        resparams.ParametersListValue.get(i) + ") is not
        boolean as learned";
    iDSLogger.ReportEvent(event);
}
b = Characteristics.CheckResDataIsDate(
    resparams.ParametersListValue.get(i));
if (b == false) {
    IDSEvent event = new IDSEvent();
    event.EventDate = new Date();
    event.EventLevel = 2;
    event.EventInfo = "A response SOAP tag (" +
        resparams.ParametersListName.get(i) + ") value (" +
        resparams.ParametersListValue.get(i) + ") is not DATE-TIME
        as learned";
    iDSLogger.ReportEvent(event);
}
b = Characteristics.CheckResDataIsNum(
    resparams.ParametersListValue.get(i));
if (b == false) {
    IDSEvent event = new IDSEvent();
    event.EventDate = new Date();
    event.EventLevel = 2;
    event.EventInfo = "A response SOAP tag (" +
        resparams.ParametersListName.get(i) + ") value (" +
        resparams.ParametersListValue.get(i) + ") is expected
        to be numerical";
    iDSLogger.ReportEvent(event);
}
b = Characteristics.CheckResDataHasChar(
    resparams.ParametersListValue.get(i));
if (b == false) {
    IDSEvent event = new IDSEvent();
    event.EventDate = new Date();
    event.EventLevel = 2;
    event.EventInfo = "A response SOAP tag (" +
```

```
                resparams.ParametersListName.get(i) + ") value (" +
                resparams.ParametersListValue.get(i) + ") contained
                unexpected characters";
            iDSLogger.ReportEvent(event);
        }
        b = Characteristics.CheckResNamesCountRange(
            resparams.ParametersListName.get(i), resparams);
        if (b == false) {
            IDSEvent event = new IDSEvent();
            event.EventDate = new Date();
            event.EventLevel = 4;
            inner:
            for (int n = 0; n <
                Characteristics.ResponseNamesCountRange[0].length; n++) {
                if (Characteristics.ResponseNamesCountRange[0][n].
                    equalsIgnoreCase(resparams.ParametersListName.get(i))) {
                    int min =
                    Integer.valueOf(Characteristics.
                        ResponseNamesCountRange[1][n]);
                    int max =
                    Integer.valueOf(Characteristics.
                        ResponseNamesCountRange[2][n]);
                    event.EventInfo = "A response SOAP tag frequency (" +
                        resparams.ParametersListName.get(i) + ") is not
                        within the learned range (" + min + ","
                        + max + ")";
                    iDSLogger.ReportEvent(event);
                    break inner;
                }
            }
        }
    }
    return report;
}
```

C.3 Checking Request/Response Dependencies

This function is responsible for initiating the process of checking whether the captured response can/cannot/may be preceded by the requests that resulted in this response.

```
public static void InvestigateDependencies(ParametersList reqparams,
   ParametersList resparams, ServiceCharacteristics Characteristics,
   IDSLogger iDSLogger) {
   boolean ANDtable[][] = Characteristics.ANDTable;
   boolean ORtable[][] = Characteristics.ORTable;
   int Lreq = 0;
   try {
      if (reqparams != null) {
         Lreq = reqparams.ParametersListName.size();
      }
   } catch (Exception e) {
      e.printStackTrace();
   }
   int Lres = 0;
   try {
      if (resparams != null) {
         Lres = resparams.ParametersListName.size();
      }
   } catch (Exception e) {
      e.printStackTrace();
   }
   for(int i = 0; i < Lres; i++)
   {
      try {
         String res = resparams.ParametersListName.get(i);
         int ires = GetIndexOf(Characteristics.ResTagsNames, res);
         for (int j = 0; j < Lreq; j++) {
            boolean bOR = false;
            String req = null;
            try {
               req = reqparams.ParametersListName.get(j);
               int ireq = GetIndexOf(
                  Characteristics.ReqTagsNames, req);
               boolean bAND = false;
               try {
                  if (ireq == -1)
                  {
                     System.out.println("-1");
```

```
                    }
                    bAND = ANDtable[ireq][ires];
                    // Must be preceeded by req
                } catch (Exception e) {
                    e.printStackTrace();
                }
                bOR = !ORtable[ireq][ires];
                // Cannot be preceeded by req
                if (bAND == false) {
                    IDSEvent event = new IDSEvent();
                    event.EventDate = new Date();
                    event.EventLevel = 4;
                    event.EventInfo = "Response tag (" + res + ")
                        must be preceeded by request (" + req + ")";
                    iDSLogger.ReportEvent(event);
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
            try {
                if (bOR == true) {
                    IDSEvent event = new IDSEvent();
                    event.EventDate = new Date();
                    event.EventLevel = 4;
                    event.EventInfo = "Response tag (" + res + ")
                        cannot be preceeded by request (" + req + ")";
                    iDSLogger.ReportEvent(event);
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
  }
}
```

BIBLIOGRAPHY

[1] Michael Stal. Web services: Beyond component-based computing. *Communications of the ACM*, 45(10):71–76, 2002.

[2] Ramarao Kanneganti and Prasad A Chodavarapu. *SOA Security*. Manning Publications, January 2008.

[3] Chris Peltz. Web services orchestration and choreography. *Computer*, 36(10):46–52, October 2003.

[4] W3C Working Group. Web services glossary, February 2004.

[5] David Sprott and Lawrence Wilkes. Understanding service oriented architecture. *The Architecture Journal*, January 2004.

[6] Gil Long and Mamdouh Ibrahim. Service-oriented architecture and enterprise architecture part 1. *Published at http://www.ibm.com/developerworks/library/wssoa-enterprise1/index.html*, April 2007.

[7] Michael N.Huhns and Munindar P.Singh. Service-oriented computing: Key concepts and principles. *IEEE Internet Computing*, 9(1):75–81, January/February 2005.

[8] David Walend. Understanding service oriented architecture. *Developer Network*, November 2006.

[9] Yvonne Balzer. Improve your soa project plans. *IBM*, July 2004.

[10] Cecilia Phan. Service oriented architecture (soa) security challenges and mitigation s trategies. *Military Communications Conference (MILCOM 2007)*, pages 1–7, October 2007. IEEE Computer Society.

[11] Dipak Chopra. Security for soa and web services. *Published at https://www.sdn.sap.com/irj/servlet/prt/portal/prtroot/docs /library/uuid/512de490-0201-0010-ffb4-8bd1620b2386*, December 2004.

[12] Robert Bunge, Sam Chung, Barbara Endicott Popovsky, and Don McLane. An operational framework for service-oriented architecture network security. *Proceedings of the 41st Hawaii International Conference on System Sciences*, 2008.

[13] Fernandez E. B. and Delessy N. Using patterns to understand and compare web services security products and standards. *Proceedings of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services(AICT/ICIW)*, pages 157–157, 2006.

[14] Skallka C. and Wang X. Trust by verify: Authorization for web services. *ACM Workshop on Secure Web Services*, pages 47–55, 2004.

[15] Yuan E. and Tong J. Attributed based access control (abac) for web services. *Proceedings of the 2005 IEEE International Conference on Web Services (ICWS'05)*, 00:561–569, 2005.

[16] Yuan E. and Tong J. Web services security: What's required to secure a service- oriented architecture. *An Oracle White Paper*, October 2006.

[17] Harold Lockhart. Demystifying security standards. *Published at http://dev2dev.bea.com/pub/a/2005/10/security_standards.html*, October 2005.

[18] Rich Cannings, Himanshu Dwivedi, and Zane Lackey. *Hacking Exposed Web 2.0 Security Secrets and Solutions*. McGraw-Hill, 2008. ISBN: 0071494618.

[19] Meiko Jensen, Niel Gruschka, Ralph Herkenhoner, and Nobert Luttenberger. Soa and web services: New technologies, new standards - new attacks. In *Proceedings of the Fifth European Conference on Web Services*, pages 35–44, Washington, DC, USA, 2007. IEEE Computer Society.

[20] D.Bell and L.LaPadula. Secure computer systems: Mathematical foundations. *Technical Report MTR*, 1(2547), March 1973.

[21] D.Bell and L.LaPadula. Secure computer system: Unified exposition and multics interpretaion. *Technical Report MTR*, 1(2997), March 1975.

[22] Matt Bishop. *Computer Security: Art and Science*. Adison Wesley, Boston, MA, 2003.

[23] Majd Al-kofahi, Su Chang, and Thomas E.Daniels. Scwim an integrity model for soa networks. In *IEEE International Conference on Web Services*, pages 675–682. IEEE Computer Society, 2008.

[24] David R. Wilson David D. Clark. A comparison of commercial and military computer security policies. *proceedings of the 1987 IEEE Symposium on Security and Privacy*, pages 148–194, April 1987.

[25] K.P. Eswaran, J.N. Gray, R.A. Lorie, and I.L. Traiger. The notions of consistency and predicate locks in a database system. *Communications of the ACM*, 19(11):624–633, November 1976.

[26] Mohammad Alrifai, Peter Dolog, and Wolfgang Nejdl. Transactions concurrency control in web service environment. *Proceedings of the European Conference on Web Services (ECOWS'06)*, pages 109–118, December 2006.

[27] Mohammad Alrifai, Peter Dolog, and Wolfgang Nejdl. Decentralized coordination of transactional processes in peer to peer environments. *Proceedings of the 14th ACM International Conference on Information and Knowledge Management (CIKM 2005)*, pages 36–43, November 2005.

[28] S. Choi, H. Jang, H. Kim, J. Kim, S. Kim, J. Song, and Y. Lee. Maintaining consistency under isolation relaxation of web services transactions. *Proceedings of the WISE 2005*, November 2005.

[29] Peter Budny, Srihari Govindharaj, and Karsten Schwan. Worldtravel: A testbed for service-oriented applications. In *Proceedings of the 6th International Conference on Service-Oriented Computing*, pages 438–452, Sydney, December 2008. Australia.

[30] Arun Gupta. http://java.net/projects/wsmonitor/, January 2011.

[31] Steven A.Hofmeyr, Stephanie Forrest, and Anil Somayaji. Intrusion detection using sequences of system calls. *IBM*, July 1997.

[32] Christina Warrender, Stephanie Forrest, and Barak Pearlmutter. Detecting intrusions using system calls: Alternative data models. *Journal of Computer Security*, 6(3):151–180, August 1998.

[33] Stephanie Forrest, Steven A.Hofmeyr, Anil Somayaji, and T. A Logstaff. A sense of self for unix process. *Proceedings of 1996 IEEE Symposium on Computer Security and Privacy*, pages 120–128, 1996.

[34] Yihua Liao and V. Rao Vemuri. Using text categorization techniques for intrusion detection. *Proceedings of the 11th USENIX Security Symposium*, pages 51–59, August 2002.

[35] Feng Pan and Weinong Wang. Anomaly detection based on the regularity of normal behavior. *1st International Symposium on Systems and Control in Aerospace and Astronautics (ISSCAA)*, page 6, January 2006.

[36] Darren Mutz, Fredrik Valeur, and Giovanni Vigna. Anomalous system call detection. *ACM Transactions on Information and System Security*, 9(1):61–93, February 2006.

[37] Christopher Kr Ugel, Thomas Toth, and Engin Kirda. Service specific anomaly detection for network intrusion detection. *Proceedings of the 2002 ACM symposium on Applied computing*, pages 201–208, 2002.

[38] P. Uppuluri and R. Sekar. Experiences with specification-based intrusion detection. *Proceedings of the 4th International Symposium on Recent advances in intrusion detection (RAID 2001)*, pages 172–189, 2001.