AN APPROACH TO IMPLEMENT SECURITY IN SERVICE ORIENTED ARCHITECTURE USING DECEPTION TECHNIQUE

By

SHAH MD. EMRUL ISLAM

Master of Science in
Information and Communication Technology
Bangladesh University of Engineering and Technology
Dhaka, Bangladesh
2004

Bachelor of Science in
Computer Science & Engineering
Ahsanullah University of Science and Technology
Dhaka, Bangladesh
2002

Submitted to the Faculty of the Graduate College
of the Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May, 2008

AN APPROACH TO IMPLEMENT SECURITY IN
SERVICE ORIENTED ARCHITECTURE
USING DECEPTION TECHNIQUE

Thesis Approved:

Johnson Thomas

Thesis Adviser

Xiaolin Li

Venkatesh Sarangan

A. Gordon Emslie

Dean of the Graduate College

ACKNOWLEDGEMENTS

First of all I thank almighty Allah for giving me the opportunity to complete this work on time. I thank my parents who brought me here in this world and raised me up with great love. I also express my gratitude to my family members and my friends for their inspiration.

My earnest thanks are due to my adviser Dr. Johnson Thomas, without his motivation, encouragement and support; this thesis effort would not have been possible. I appreciate his time and effort put forth towards my thesis.

In addition to this I am grateful to my committee members Dr. Venkatesh Sarangan and Dr. Xiaolin Li for their valuable suggestions as committee members. Their guidance helped me a lot to converge on the topic. Finally I am grateful to William G.J. Halfond of Georgia Institute of Technology for his help on the SQL Injection Attack implementation.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Service Oriented Architecture [1,12,13,14,15,16,19,20,21,22,23,24,25,26,27,28,32] (SOA) is an architectural style whose goal is to achieve loose coupling among interacting software agents. A service is the unit of work done by a service provider to achieve desired results for a service consumer. Web services standards and the SOA provide help toward opening IT infrastructures by providing a uniform way to expose the functionalities through standards like Web Service Description Language (WSDL) [8] and Simple Object Access Protocol (SOAP) [5], and to discover web services through standards like Universal Description Discovery and Integration (UDDI) [17,18]. SOA differs in many ways from many other distributed technologies. However, as SOA is becoming widely popular, it is becoming an attractive target for attackers and hackers. SOA works on top of the internet and therefore it is more open to attackers. SOA has the ability to easily communicate with customers, vendors and other systems outside the company. Therefore there are some vulnerability from the communication between service consumers and service providers.

Detecting an attack and refusing the request of an attacker is usually the first line of defense to SOA attacks. Such an approach helps prevent an attacker from breaking into

the system. However, such an approach does not prevent or deter the attacker from trying again using a slightly different mechanism. In most of cases the attacker tries to automate the attack by writing codes which is then executed hundreds and thousands of times to break into the system.

Many works have been done to secure SOA and such work continues. Most if not all of the works aim to detect an attack and thereby prevent an attacker from accessing the system. In this thesis we proposed a different model to counter attacks. We use deception as a mechanism for deterring an attacker. We looked at an SOA organization and focused on the security issues that are relevant to SOA. Although deception cannot guarantee that an attack can be prevented, it slows the attacker down. In a best case scenario, the attacker may not detect he is being fooled. Even if an attacker becomes aware of the deception, it will have used up some of his resources and time. Deception also allows us to study the behavior of the attacker. This information can be used to design better security models. Deception also buys time for the defender. While the attacker is being deceived, the defender can enforce his security by strengthening the keys for example. Hence our proposed approach has many advantages. Deception works on top of intrusion detection because only after detecting an intrusion, can deception be used. As far as we are aware, no-one else has applied deception to SOA.

The proposed method is thus developed to help reduce attacks on SOA services. The objective of this thesis is to apply deception to SOA attack and determine the effectiveness of deception against attacks on web services.

In chapter 2 we study the concepts and technical details of SOA as well as the security issues relevant to SOA security. Chapter 3 presents in detail the proposed approach. We describe the specific attacks that we implemented for our work. In Chapter 4 we specify the implementation details and the findings from our work; we also analyze the benefits of this method. Chapter 5 concludes the work and identifies future possible work.

CHAPTER 2

LITERATURE REVIEW

2.1 Service Oriented Architecture:

Service Oriented Architecture (SOA) is an architectural style whose goal is to achieve loose coupling among interacting software agents. A service is a small or large unit of work done by a service provider to achieve the desired output for a service consumer. Both service provider and consumer are roles played by software agents from their respective sides on behalf of their owners. The idea of SOA departs significantly from that of object oriented programming (OOP), which suggests that data and its processing, should be bound together. An application's business logic or individual functions are basically modularized and represented as services for consumer/client applications. The service interface is independent of the implementation. Application programmers or system integrators can build applications by composing one or more services without knowing any detail about the services' underlying implementations. For example, a service can be implemented either in .Net or J2EE or any other platform, and the application consuming the service can be on a different platform or language. However, this would not hamper the normal operation of the process.

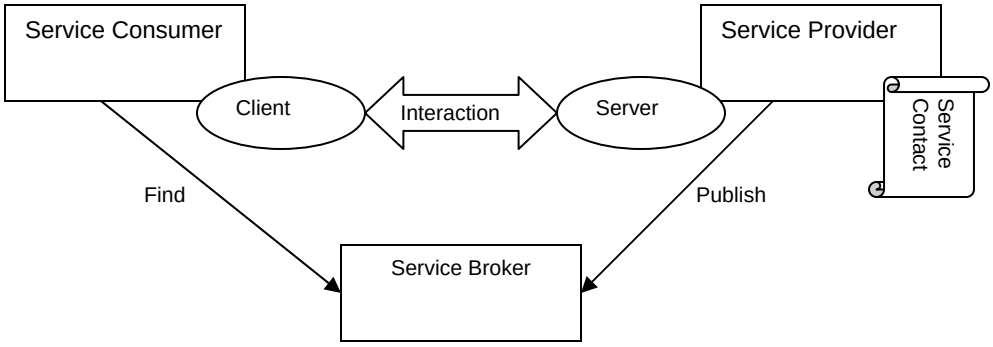

Figure 2.1: Service Oriented Architecture

2.2 Benefits of Service Oriented Architecture (SOA)

The concept of SOA is not new; SOA is different from existing distributed technologies. Almost all of the vendors have accepted it and there is an application or platform suite that enables SOA. SOA has a set of standards that brings better reusability of existing assets or investments in the enterprise and allows somebody to create applications that can be built on top of new and existing applications.

SOA has enabled changes to applications while keeping clients or service consumers inaccessible from evolutionary changes that take place in the implementation of a new or existing service. It enables upgrading individual services or services consumers. However, it is not necessary to completely rewrite the code of an application or keep an existing system that is no longer required for the business. Lastly, SOA provides enterprises better elasticity for building applications and business processes in an agile

way by influencing existing application infrastructure to compose new services which will be used for the business.

SOA creates firm connections with the customers and the suppliers of a business by allowing the creation of dynamic applications and business services which are available to external customers and suppliers. It increases the customer/partner satisfaction.

SOA provides improved business decision making by combining access to business services and information into compound business applications which are dynamic as well. The decision makers get precise and comprehensive information. It also increases the flexibility of accessing the information in the form that meets their requirements.

SOA gives greater employee productivity by providing smooth access to the systems and information. It enables business process improvement and because of that businesses can have greater employee productivity. Employees can address the important, value-added processes rather than having to be traditional to the limitations and restrictions of the underlying IT systems.

2.3 List of Markup Languages for Web Services:

The following languages are modern day standards for SOA/ Web Services markup languages:

Web Services Description Language (WSDL)

Web Services Conversation Language (WSCL)

Web template

Web interface languages

Web Service Modeling Language

XML Interface for Network Services (XINS)

Web Services Flow Language (WSFL)

Web Service-Metadata Exchange

Representational State Transfer (REST)

XML-RPC - XML Remote Procedure Call

2.4 Web Services Description Language:

The Web Services Description Language (WSDL) is an XML-based language that provides a model for describing Web services. It is an XML-based service description on communication using web services. WSDL defines services as collections of network endpoints or ports in other language.

The theoretical definition of ports and messages is divided from their concrete use or instance, allowing the reclaiming of these definitions. A port is an association of network addresses with a reusable binding, and a collection of ports define a service. Messages are theoretical descriptions of the data which is exchanged, and port types are the theoretical collections of supported operations. The existing protocol and data format specifications

for a particular port type constitutes a reusable binding, where the messages and operations are then bound to an existing network protocol and message format. WSDL thus explains the public interface to the web service.

WSDL is often used in amalgamation with XML Schema and SOAP for providing web services over the Internet. A client program which is connecting to a web service is able to read the WSDL to determine what functions are offered by the server. Any special data types that are used are embedded in the WSDL file in the form of XML Schema. The client can then use SOAP for calling one of the functions listed in the WSDL.

2.5 Universal Description, Discovery and Integration (UDDI):

Universal Description, Discovery and Integration (UDDI), which is a key component of SOA, is a platform-independent, XML-based registry for the universal business systems to list them on the Internet. UDDI which is an open industry scheme which is sponsored by OASIS, enables business systems to circulate service listings and discover each other and define how the services or software applications will interact over the Internet. A UDDI business registration consists of three components:

White Pages — keeps address, contact, and known identifiers

Yellow Pages — keeps industrial classifications on the basis of standard taxonomies

Green Pages — keeps technical information about a services uncovered by the business

UDDI is a core language designed to provide access to WSDL documents describing the protocol bindings and message formats that are required to interact with the web services listed in its directory.

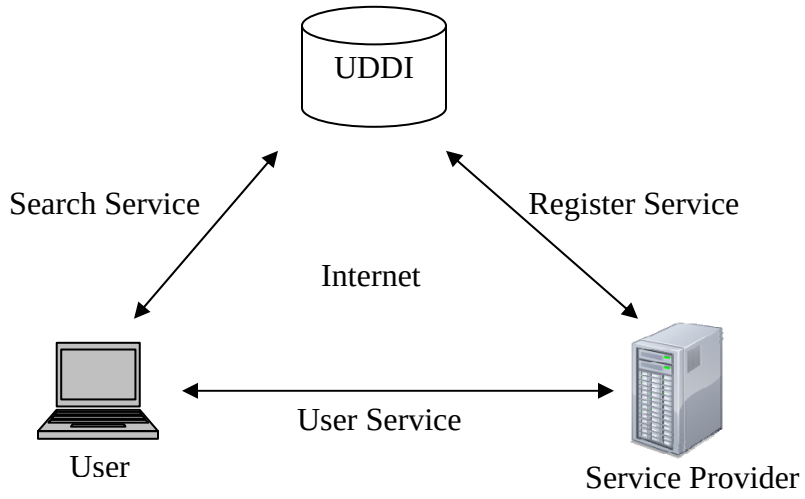

Figure 2.2: Organization of UDDI

2.6 The Web Service Protocol Stack:

The Web service protocol stack is a collection of computer networking protocols that are used to locate, define, implement, and make Web services interact with each other. The Web service protocol stack contains the following:

Service Transport Protocol:

This protocol is responsible for the transportation of messages between network applications and it includes protocols such as Hyper Text Transport Protocol (HTTP),

Simple Mail Transfer Protocol (SMTP), File Transfer Protocol (FTP) and Blocks Extensible Exchange Protocol (BEEP).

XML Messaging Protocol:

This is responsible for encoding messages in a common XML format understandable at any end of a network connection. Examples of this protocols are XML-RPC, WS-Addressing, SOAP and REST.

Service Description Protocol:

Service Description Protocol is used for describing the public interface to a specific web service. The WSDL interface format is typically used for this purpose.

Service Discovery Protocol:

This consolidates services into a common registry such that web services can publish their location and description which makes it easy to discover what services are offered on the network. As of now UDDI is normally used for service discovery.

2.7 Simple Object Access Protocol

SOAP is a protocol for exchanging XML-based messages over computer networks. It normally uses HTTP/HTTPS. SOAP usually forms the foundation layer of the Web services stack. It provides a basic messaging oriented structure that more abstract layers can build on. There are many kinds of messaging patterns in SOAP. Most of them are

Remote Procedure Call (RPC) patterns, where one network node sends a request message to another node (client server technology), and the server immediately sends a response message to the client. SOAP is the descendant of XML-RPC although it uses its transport and interaction impartiality and the header/body/envelope from elsewhere. The SOAP specification is currently maintained by the XML Protocol Working Group of the WWW Consortium.

2.8 Web Services Security:

Web Services Security [6, 30, 31, 33] or WS Security in a Service oriented Architecture is a vital feature for technologies to be completely adopted by the industry.

Remote attacks can be a key problem on a WS. These attacks are initiated from various sources external to the business systems. In this regards the firewalls perform well by blocking incoming traffic to services that should not be uncovered to the outside network.

Nevertheless, hackers always break known security implementations for an enterprise. That’s why the deception method works very well in addition to the prevention method. Honeypot [2, 3] is a very good technique for trapping a hacker which is mostly used right after network intrusion detection system.

As SOA is comparatively a novel standard in today's computer world therefore there is still a lack of worldwide approaches that offers a systematic development for constructing robust security architectures for SOA/WS-based systems.

2.9 Security Implementation Aspects:

Security implementation of SOA/web services includes many aspects. In order to maintain the robust security of the web services these factors are very important. Some of these aspects are as following.

Authentication: Source of the request.

Authorization: Is the authenticated source entitled to access or not.

Privacy: Is it ok or not to release personally exclusive information to everybody.

Integrity: Is it the original message, or tampered message.

Confidentiality: Is the information open/close to everybody while it is in transit.

Availability: Is it vulnerable to a denial of service [29] attack.

Non-Repudiation: To ensure that a contract cannot later be denied by either of the parties involved in terms of sending/receiving message.

Policy Administration: Ease to apply or change a security policy rule.

Audit: Keeping track of the transaction occurred.

2.10 Acceptance of New Web Security Standard for the Industry:

As of now web service security standards are developed and maintained by several different organizations such as W3C, OASIS (Organization for the Advancement of Structured Information Standards), Liberty Alliance, and an industry forum headed by Microsoft and IBM. Industry forum specifications for web security are submitted to W3C/OASIS for maintenance and acceptance as a standard. W3C/OASIS standards adoption committees provide forums that unite specific industries or communities of users, governments, vendors, industry groups, and other standards bodies. These committees evaluate existing standards, articulate requirements, identify gaps, recognize overlaps, publish guidelines, and promote interoperability. They provide input to W3C/OASIS technical committees that develop pertinent specifications, and they recommend new efforts where needed.

2.11 Common Threats to Web Services

There many kinds of threats related to web services [34]. Based on the security aspects of Web Services the threats could be classified into following categories.

2.11.1 Authentication

2.11.1.1 Brute Force:

A Brute Force attack is a programmed process of trial and error used to guess a person's username, password, cryptographic key or any other security token.

2.11.1.2 Insufficient Authentication:

Insufficient Authentication occurs when a web service/ web site permits an attacker to access sensitive content or functionality without proper authentication.

2.11.1.3 Weak Password Recovery Validation:

This occurs when a web site allows an attacker to illegally obtain, change or recover another user's password. This is a process of guessing the unique value that identifies a particular session or a user. The consequences allow attackers to issue requests with the compromised user's privileges.

2.11.2 Authorization

2.11.2.1 Credential/Session Prediction:

Credential/Session Prediction is a technique of hijacking a web site user.

2.11.2.2 Insufficient Authorization:

This happens when a web service /web site permits access to sensitive functionality that should require different access control restrictions.

2.11.2.3 Insufficient Session Expiration:

Insufficient Session Expiration happens when a web site allows an attacker to reuse old session credentials for authorization.

2.11.2.4 Session Fixation:

This is an attack technique which forces a user's session ID to an unambiguous value.

2.11.3 Client-side Attacks

2.11.3.1 Content Spoofing:

Content Spoofing is an attack practice which is used to trap a user into assuming that certain content appearing on a web site is genuine and not from an external source.

2.11.3.2 Cross-site Scripting:

Cross-site Scripting is an attack technique which forces a web service / web site to echo attacker-supplied executable code, which loads at a user's end.

2.11.4 Command Execution

2.11.4.1 Buffer Overflow:

These are attacks that change the flow of an application by overwriting parts of the memory.

2.11.4.2 Format String Attack:

This changes the regular flow of an application by using string formatting library which features to access other memory space.

2.11.4.3 LDAP Injection:

LDAP Injection is an attack practice that is used to take advantage of web services / web sites that construct LDAP statements from user-supplied input data.

2.11.4.4 OS Commanding:

This is an attack technique used to exploit web services / web sites by executing Operating System commands by the manipulation of application input.

2.11.4.5 SQL Injection:

This attack technique is used to take advantage of web services/ web sites that construct SQL statements from user-supplied input.

2.11.4.6 SSI Injection:

This is a server-side exploit technique which allows an attacker to send code into a web service / web application, which will later be executed locally by the web server.

2.11.4.7 XPath Injection:

This is an attack practice used to exploit web services / web sites that construct XPath queries from user-supplied input.

2.11.5 Information Disclosure

2.11.5.1 Directory Indexing:

Automatic directory listing/indexing is a web server function which lists all of the files within a requested directory if the normal base file is non-operational.

2.11.5.2 Information Leakage:

In this situation a web site reveals sensitive data, such as developer comments or error messages, which may help an attacker in exploiting the system information.

2.11.5.3 Path Traversal:

This technique forces access to files, directories, and commands that reside outside the web document root directory.

2.11.5.4 Predictable Resource Location

This is an attack technique which is used to expose hidden web site content and functionality.

2.11.6 Logical Attacks

2.11.6.1 Abuse of Functionality:

This is an attack technique that uses a web services' / website's own features and functionality to use, defraud, or get around access controls mechanisms.

2.11.6.2 Denial of Service:

This is an attack technique with the intent of preventing a web service / web site from serving normal user activities.

2.11.6.3 Insufficient Anti-automation:

This happens when a web service/ web site permits an attacker to automate a process that is supposed to be performed manually.

2.11.6.4 Insufficient Process Validation:

This happens when a web site allows an attacker to bypass the planned flow control of an application.

2.12 Related Works

Kevin et al developed a system called OpenFire [3] which is derived from the concept of a Honeypot and used in fooling hackers against network attacks. It uses deception to interfere with the reconnaissance phase. Unlike traditional firewalls, instead of blocking unwanted traffic, it accepts all traffic, forwarding unwanted messages to a cluster of decoy machines. To the outside, all ports and all IP addresses appear open in an OpenFire network. However, this work was based on deception at the network level and not on SOA.

A lot of work has been done on SOA security. Some of these are outlined below. However, none of them involved deception and are therefore quite different from our work.

Karthikeyan et al of Microsoft Corporation designed an advisor for web service security policies [4] as a part of a research project of Microsoft. They identified some common security vulnerabilities found during security reviews of web services with policy-driven security. They described the design of an advisor for web services security configurations; they claimed it to be the first tool both to identify such vulnerabilities automatically and to offer remedial advice.

Mohammad et al developed a method for secured SOAP message exchange technique [5] which is very important for secured communication in SOA. They provided a solution which is based on the usage of message structure information for preservation of message integrity. They also discussed the integrity feature of a SOAP account.

Takeshi et al tried to figure out the necessary requirements for the configuration of security in SOA [6]. They came up with refining security requirements from business to technology, leveraging the concepts of Service-Oriented Architecture (SOA) and Model-Driven Architecture (MDA).

Liang et al worked on developing a method for password based authentication key exchange for web services [7]. They discussed an implementation of an authenticated key exchange method rendered on message primitives defined in the WS-Trust and WS-SecureConversation specifications.

Carlisle et al outlined a framework for implementing security for Web Services by extending UDDI and WSDL [8]. They presented the PWSSec (Process for Web Services Security) process that is composed of three stages, WSSecReq (Web Services Security Requirements), WSSecArch (Web Services Security Architecture) and WSSecTech (Web Services Security Technologies) that accomplishes the mentioned activities, respectively. They also provided a thorough explanation of the WSSecArch (Web Services Security Stage) stage intended to design the web services-based security architecture.

Keromytis, A. et al presented SABER (Survivability Architecture: Block, Evade, React), a proposed survivability architecture [9] that blocks, evades and reacts to a variety of attacks by using several security and survivability mechanisms in an automated and coordinated fashion. It integrates several different technologies in an attempt to provide a unified framework for responding to the wide range of attacks malicious insiders and outsiders can launch. This was a kind of coordinated multi-layer approach which is capable of defending against attacks targeted at various levels of the network stack, such as congestion-based DoS attacks, software-based DoS or code-injection attacks, and others.

Hosamani, M. et al proposed an extension of the service-oriented architecture [10] that provides trustworthiness means for clients to specify, service brokers to verify, and service implementations facilities. They discussed a prototype implementation of that architecture and reported preliminary results that demonstrate the potential practical value of the proposed architecture in real-world software applications.

Mecella, M. et al presented a framework for enforcing access control in conversation-based Web services [11]. They considered the conversational nature of Web services and worked towards a unique solution.

CHAPTER 3

PROPOSED APPROACH

As we saw in the previous chapter a lot of work has been done on security implementation of SOA services, but none of them used deception in their work. It is true that deception has some overheads and it can make the system slower. We need to engage an extra database service for providing the attacker some fake data. We also need to classify the request in the entry point of the system. Although deception might not be a full proof technique for protecting a system, it can reduce the number of attacks on a system. This is done in two ways. The first way is by introducing a delay for the attacker. This will initially create a significant service time difference between a real request and an attack. The second way is to provide some data which is not real. It would be initially hard for an attacker to recognize that the data is false. The attacker may finally figure out that he/she has been fooled and would try a different technique of attack on the web service. However, this incurs a delay and also uses the attacker's resources, thus making the system more secure.

The implementation method is based on intrusion detection on servers where a specific web service is running. If an intrusion is detected the request for the service is classified as a “false request/ unauthenticated request”. Otherwise it is classified as a “true request”.

There are two database services that run for serving the requests of the service consumers. One database service is the real one and another is a copy of the real database service which is designed to provide fake data to the false request and thus acts as a Honeypot. However, based on the decision taken by the intrusion detection system (implemented in the web server) the false request is redirected to the false database service after a random delay which is running on an ordinary server so as not to provide better service quality for the attackers and thus save money for the organization.

We used Microsoft Visual Studio .NET for developing the web services for the service oriented architecture setup, Microsoft Internet Information Server .NET for deploying the services, and Microsoft SQL Server 2005 / Microsoft Access 2003 as the database server. All the software that we used to deploy the setup was available through the MSDN Academic Alliance (MSDNAA) program.

The intention behind this is to kill the time of the hacker and not to let him realize that his request has been refused. Hence, the hacker would think that he is working in the real world on a true database and will not try to break the system immediately. This will reduce the number of attacks to the web service running in the SOA of any enterprise.

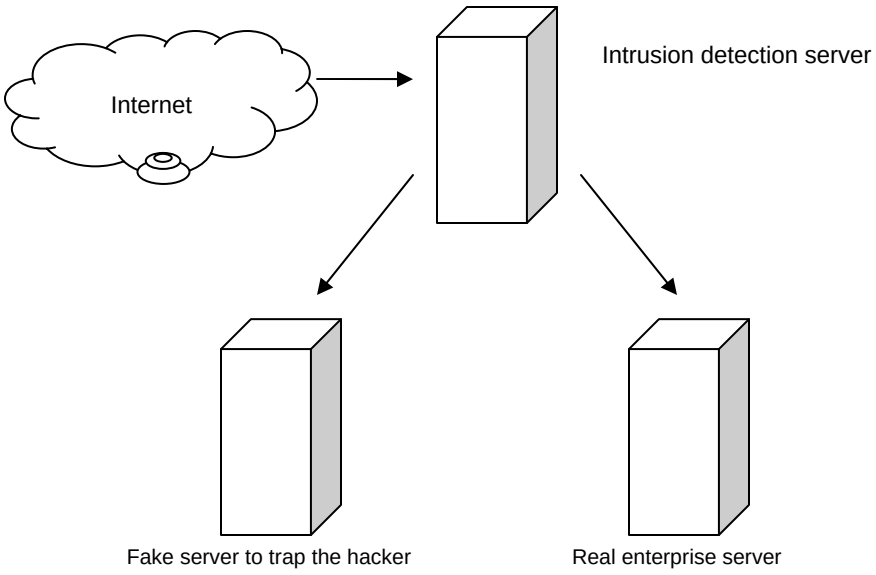

Figure 3.1: Organization of the servers

3.1 Implemented Attack Models:

When we started our work we tried of implement the system using a pure network intrusion detection system where the hacker or requester would be directed to the original service / Honeypot based on the intrusion detection system's decision on the request. However, after some initial work we realized that the network intrusion prevention will not be sufficient for our work because a network intrusion detector would not be able to detect attacks specific to SOA. Therefore we decided to study about threats that are specific to web services only.

According to the classification of threats that we studied in chapter 2, we have chosen three categories of attacks to implement in our work. We picked up one attack from each category and created an attack model for them. This selection was not based on any specific criteria but we just wanted to implement more than one kind of attacks for our work. The attack categories we selected are Brute Force Attack (Authentication category), SQL Injection Attack (Command Execution Category) and Insufficient Authorization Attack (Authorization category).

3.1.1 SQL Injection Attack

This is an attack technique targeted at web services that construct Structured Query Language (SQL) queries from user-supplied input data. Almost all of the database enabled applications can be accessed using a SQL statement which has both ANSI and ISO standards.

Nevertheless, many database products supporting SQL implement the basic standard and add their own standards as well. For example Oracle database uses the basic SQL standard and in addition they implement SQL Plus which is their proprietary extension. Web services use user-supplied input data to create their custom SQL statements for dynamic web page requests. However, when a web service fails to disinfect user-supplied input, it is likely for an attacker to change the construction of the backend SQL statements the web service is going to use. So, when an attacker can modify a SQL statement, the process will run with the same permissions in the background as the

component that executed the command. Therefore the impact of this attack can allow attackers to gain complete control of the underlying database.

Example

The simplest example of an SQL Injection can as shown below:

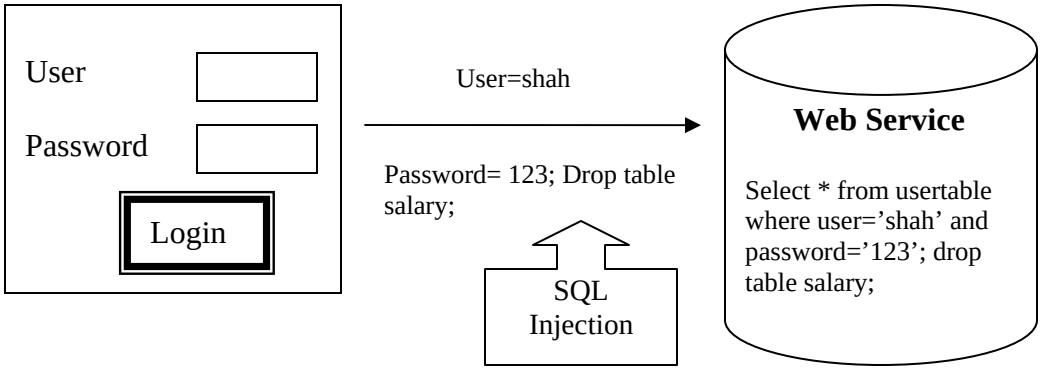

Figure 3.2: A web based authentication form with SQL Injection

SQLQuery = "SELECT Username FROM Usertable WHERE Username = '" & stringUsername & "' AND Password = '" & stringPassword & "';"

In this code, the system is getting the user-input from the input form and embedding it to an SQL query which is going to be executed in the system immediately.

Suppose an attacker submits a login and password that looks like the following:

Login: ' OR "='

Password: ' OR "='

This will cause the resulting SQL query to become the following:

SELECT Username FROM Users WHERE Username = " OR "=" AND Password = " OR "="

Instead of comparing the user-supplied data with the rows in the Usertable table, the query will compare " (empty string) to " (empty string). Hence, this will return a True result and the attacker will be logged in as the first user in the Usertable table.

There are two kinds of SQL Injection (i) Normal SQL Injection and (ii) Blind SQL Injection. Normal SQL Injection attack occurs by appending a union keyword with a select statement to the parameter, the attacker can test to see if he can gain access to the database:

http:// bursar.okstate.edu/payment.php?ID=2+union+all+select+name+from+sysobjects

The SQL server then might return an error similar to this:

Microsoft OLE DB Provider for ODBC Drivers error '80040e14'

[Microsoft][ODBC SQL Server Driver][SQL Server]All queries in an SQL statement containing a UNION operator must have an equal number of expressions in their target lists.

This will tell the attacker that he must now guess the correct number of columns for the SQL statement to work.

In a Blind SQL Injection, instead of returning a database error, sometimes the server returns a customer-friendly error page telling the user that a mistake has been made. In this scenario an SQL Injection is still possible but hard to detect. However, a general way to detect a Blind SQL Injection is to put a false and true statement into the parameter value.

Executing the following request to a web site:
http://bursar.okstate.edu/payment.php?ID=2+and+0=0 should return the same web page as:
http://bursar.okstate.edu/payment.php?ID=2 because the SQL statement 'and 0=0' will be always true.

Executing the following request to a web site:
http://bursar.okstate.edu/payment.php?ID=2+and+1=0 would then cause the web site to return a friendly error or no page at all. This is because the SQL statement "and 1=0" is always false. Now once the attacker realizes that a site is susceptible to Blind SQL Injection, he can use this vulnerability more easily, in some cases, than by using normal SQL Injection.

In our approach we treated SQL injection in two ways. The first way was whether the injected SQL statement is a Data Definition Language (DDL). Even through we deceived the attacker, we didn't allow him to run DDL SQL statements on the deception database. We only allowed the SQL injection for Data Manipulation Language (DML) statements to the deception database and deceived him. In SQL injection attack modeling in our work, we did intrusion detection, prevention and finally deception against an SQL injection attack. Nevertheless, the deception delay was introduced to any kinds of SQL injected requests.

3.1.2 Brute Force Attack

A Brute Force attack is an automatic process attack based on trial and error basis which is used to guess a person's username, password, credit-card number or cryptographic key, etc.

There are numerous systems that allow the use of weak passwords or cryptographic keys, and users often choose an easy password which is easy to guess. Therefore, an attacker automatically generates thousands of incorrect guesses to search for the valid username/password. When a guessed username/password allows access to the system, the brute force attack succeeds and the attacker gains access to the account. The same trial and error technique is also applicable to guess many other authorization information. When a web service uses a weak/small key length, it is easy for an attacker to guess a correct key by testing all possible combinations.

Brute force attack has two types. One is the normal or generic brute force and the other one is the reverse brute force attack. The first one uses a single username against multiple passwords. The other one uses many usernames against a single password. Brute force techniques are very popular and often successful but they can take minutes, hours, months or years to complete.

In our approach we used only the generic brute force and not the reverse brute force approach. We didn't exclude intrusion detection and prevention but added deception as a complement to the intrusion detection and prevention approach. We detected a brute force attack by detecting the number of unsuccessful requests and the source of the request. As most systems do, we considered a request as an "attack" after three unsuccessful tries. Therefore, after three unsuccessful tries, the attacker was never allowed to work on the original database but was allowed to work on the deception database if any of its brute force attack succeeded.

3.1.3 Insufficient Authorization

This happens when a web service allows access to sensitive content or functionality that requires increased or special access control credentials. After a user is authenticated to a web service, it does not necessarily mean that he should have full access to all the contents and functionalities. However, authorization procedures are performed after authentication, clarifying what a user, service or application is allowed to do with the system.

Example:

In a business organization there are many levels of access qualifications to the information system that the business uses. As an example a web service permits a development office to connect to the head office where the human resources department employees / managers work. However an attacker may try to access confidential files through the web service from a computer which is not a human resources department computer. Therefore this access should not be granted.

In our approach to Insufficient Authorization attack we introduced a generic delay for the attackers after detecting the attack. We kept authorization records for any functionalities supported by the web service. Therefore, anybody trying to access a functionality which should not be accessed by him was treated as an attack. After the detection we redirected the attacker to the deception configuration and provided them the data they were trying to access; but those data were coming from the deception database of course.

CHAPTER 4

SIMULATION MODEL AND FINDINGS

Our objective is to find the percentage of attack reduction using our proposed approach. We implemented our approach in a service oriented system. We believe that intrusion detection, redirection to fake service, puzzling the hacker, providing fake data and misguiding him, and finally killing some time would serve as an obstacle for a hacker to break into enterprise servers. Therefore, the real service would be able to reduce some attacks.

In our setup we used Microsoft .NET technology to simulate our experiment. We created a basic intrusion detection server for the three attacks we decided to model. In our setup we had an intrusion detection service which was running on top of Internet Information Service (IIS). We had another client which was generating valid requests to the server. We also generated a series of attacks from the client end for different probability attack values where attacks were embedded with valid requests.

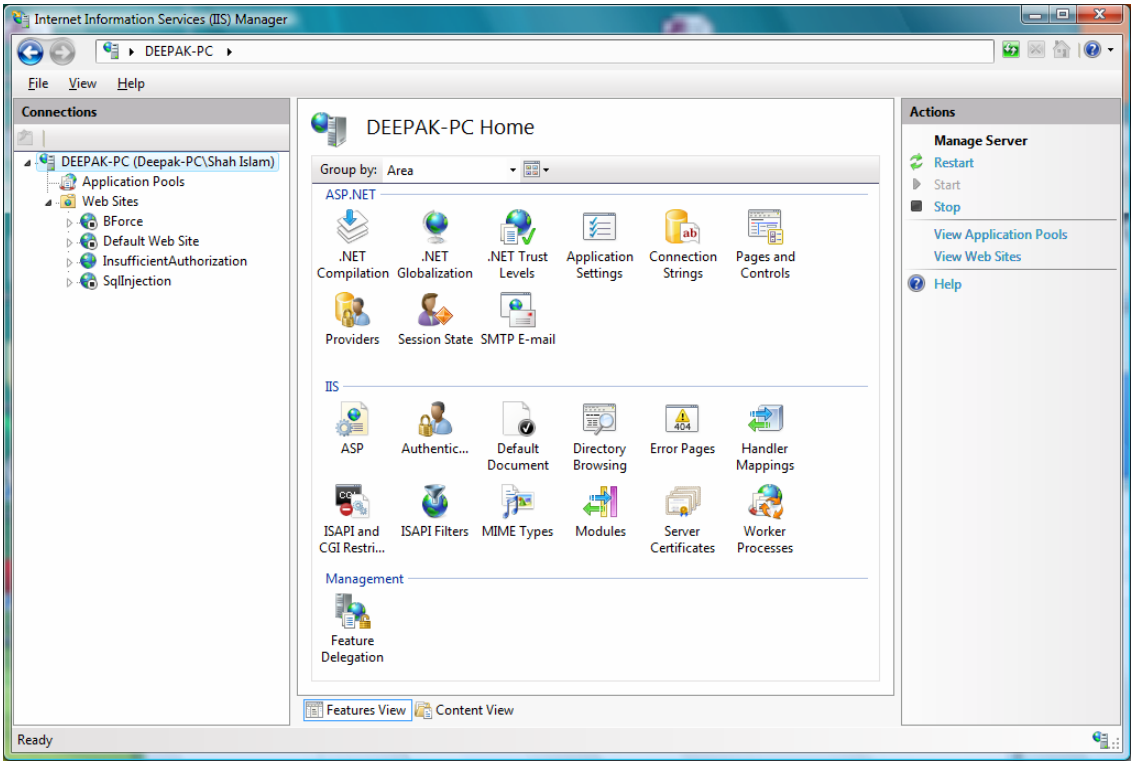

Figure 4.1: IIS Setup for the simulation

In our work we tried to simulate an SOA setup. Our implementation was divided into three tiers.

Web Server – We used Internet Information Server, Version 7.0. The web server was hosting the services on top of the application server. Figure 4.1 describes the web server organization.

Application Server – Microsoft Visual Studio Development Server 2005 was used to build the application tier of the service oriented architecture. We didn't use the concept of Enterprise Service Bus (ESB) for our setup but we deployed each service separately. Figure 4.2 shows an example of the middle tier development window.

Database Server – Microsoft SQL Server 2005 / Microsoft Access 2003 was used to host the database services for our purpose. We used a database server to host the original database for the enterprise and the Honeypot.

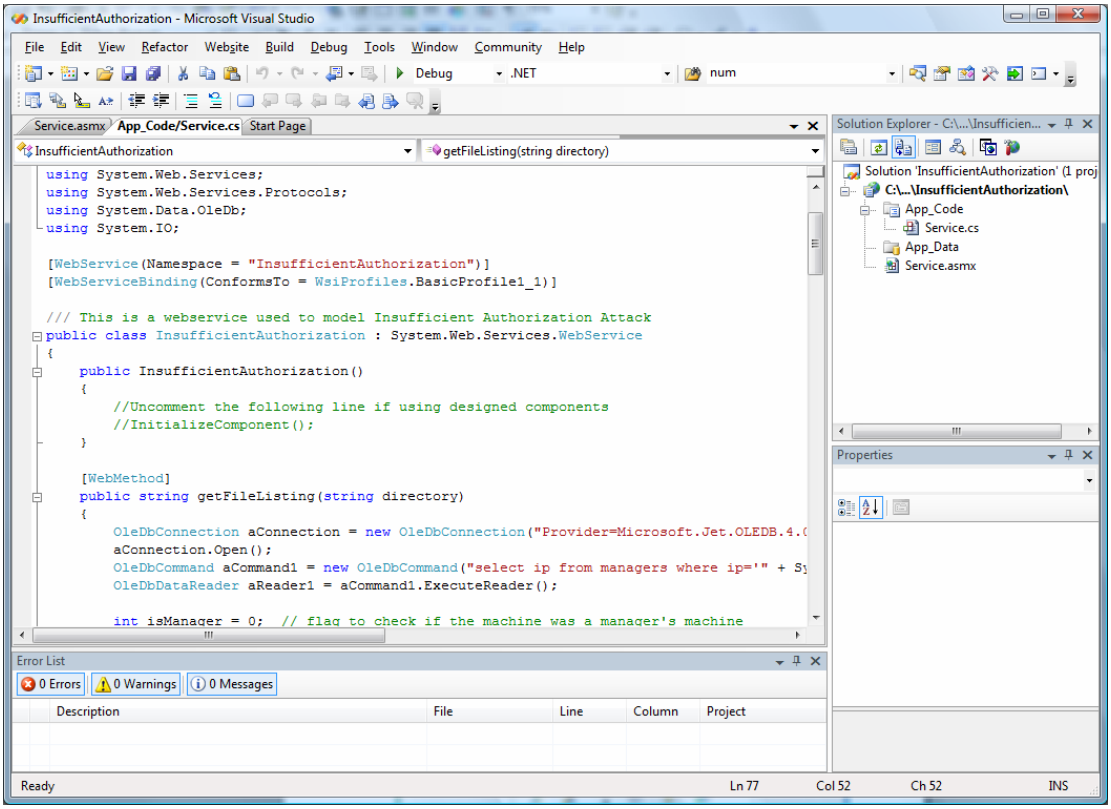

Figure 4.2: Middle-tier setup for the simulation

Service Consumer- The service consumer for the web services was deployed in a different location which we used to generate valid requests and attacks. We didn't use UDDI registry for our setup but we mapped the service consumer with the service provider by a service discovery file.

As discussed earlier that for our work we simulated three kinds of attacks.

(i) SQL Injection Attack

(ii) Brute Force Attack

(iii) Insufficient Authorization Attack

We wrote code to implement both the attack model and the intrusion model and we tested the models with a significant volume of data. From the simulation results we generated graphs that showed the delay introduced to the hacker and time of deception which prevented him from immediately attacking the web service.

The deception strategy can redirect an attacker to many different places depending on the algorithm on the web server. For example, an attacker can retrieve a false credit card number from the database (as a part of the false data given to the attacker) and eventually try to buy many things with that credit card (which will never work anyway). In our simulation we tried to deceive the hacker in one step only. For example, in the implementation of Insufficient Authorization attack model if an attacker tries to get the directory listing of a directory which he/she is not supposed to see, the system holds the attacker for a particular time delay, and eventually returns the directory listing with some file names but those are not real of course. Therefore, we calculated the delay given to the attacker in the first place when he/she is trying to get the name of the files, and not what he/she does with the filenames later.

We plotted the graphs for the data we got from our simulation results. The results came from the deception behavior we used in our intrusion detection server. For all three attack

models the server behavior was more or less the same but in the real world this might vary. However, the motto of our work is to reduce the attacks on a web service.

(i) SQL Injection Attack

We wrote our codes in ASP.NET and C# based on the following deception algorithm for SQL Injection Attack

```
while (true) //this is a service, so it will run forever
do
        request=true //first treat the request as a true request
        ddl_statement=false  //first guess that there is no DDL injection

                for i=1 to number_of_fields_in_request
                do
                        if request_field[i].contains("select from"||"group by"||"truncate
                        table"||"drop table".....) //any SQL statement is present
                        then
                                request=false   //attack
                                delay_in_seconds=random (10-100) //delay in seconds
                                pause(delay_in_seconds) //introduce a random delay

                                if request_field[i].contains("alter table"||"truncate
                                table"||"drop table".....) //any DDL statement
                                then
                                        ddl_statement=true  //DML SQL injection
                end if
                        end if
                end do

        if request=true then
                connectionstring=original_database_connection_string
        else
                connectionstring=false_database_connection_string
        end If

        connect_to_database (connectionstring)

        if ddl_statement=false
                then result=execute SQL    //DML injection on Honeypot
        else
```

```
            result="no data found"
        end if

        return result   //return result for request
end do
```

This algorithm was run on three sets of 5000 requests from a client that we setup. The requests were incessant and attacks were embedded on those requests for a probability of 0.1, 0.2 and 0.3 (Probability 0.1 means out of 10 requests one is an attack and so on). We got the results shown below from the simulation. The service time considered for any request (real request or attack) is 3 seconds. From these graphs we can clearly see that the time to deceive is directly proportional to the attack probability.

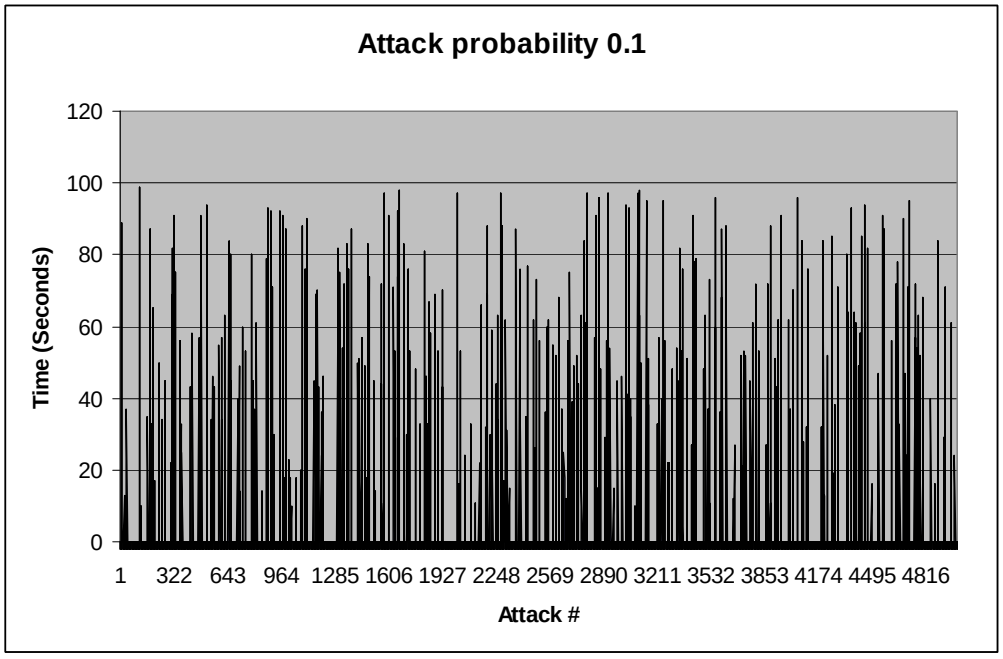

Figure 4.3: SQL Injection Attack Findings for Attack probability of 0.1

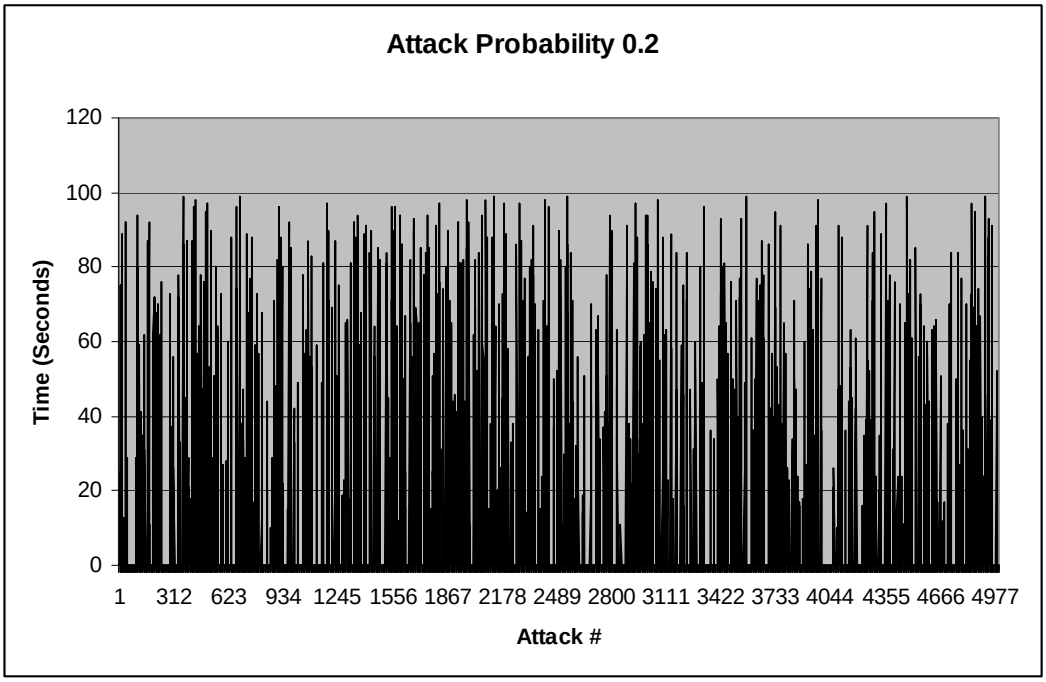

Figure 4.4: SQL Injection Attack Findings for Attack probability of 0.2

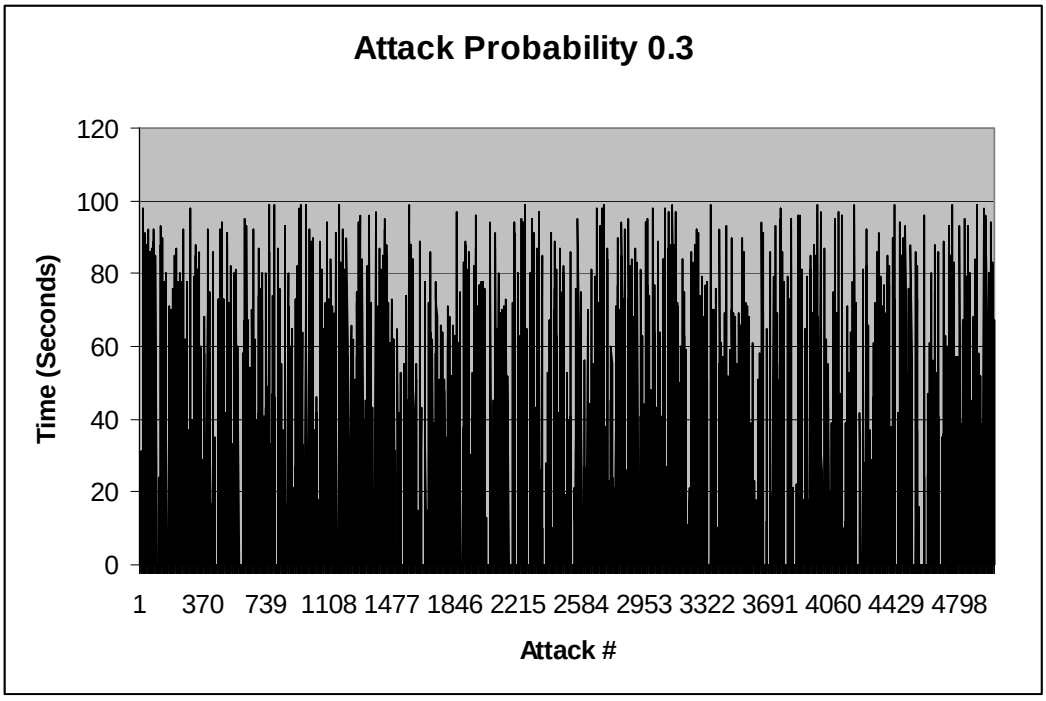

Figure 4.5: SQL Injection Attack Findings for Attack probability of 0.3

(ii) Brute Force Attack

We wrote our codes in ASP.NET and C# based on the following deception algorithm for Brute Force Attack

```
while (true)  //this is a service, so it will run forever
do
        if try>3 then   //if three unsuccessful attemps
                delay_in_seconds=random (10-100) //delay in seconds
                pause(delay_in_seconds) //introduce a random delay
                connectionstring=false_database_connection_string
        else
                connectionstring=original_database_connection_string
        end if

        if login(user,password)=unsuccessful //no data found for this user
        then
                try=try+1  //increase number of unsuccessful tries
                return "login unsuccessful"  //return a general message to the user
        else
                return "login successful"  //return a general message to the user
        end if
end do
```

This algorithm was run on two sets based on the password length. The first set had 676 requests for a permutation of two password lengths for the alphabetical characters. The second set had 17576 requests for a permutation of three password lengths for the alphabetical characters. As expected the results shown below show that for a brute force attack it could take an attacker years to try every combination if the password length is more. However, the attacker would not succeed with a brute force attack after three unsuccessful tries. As before, we considered the service time as three seconds for any request made to the web service.

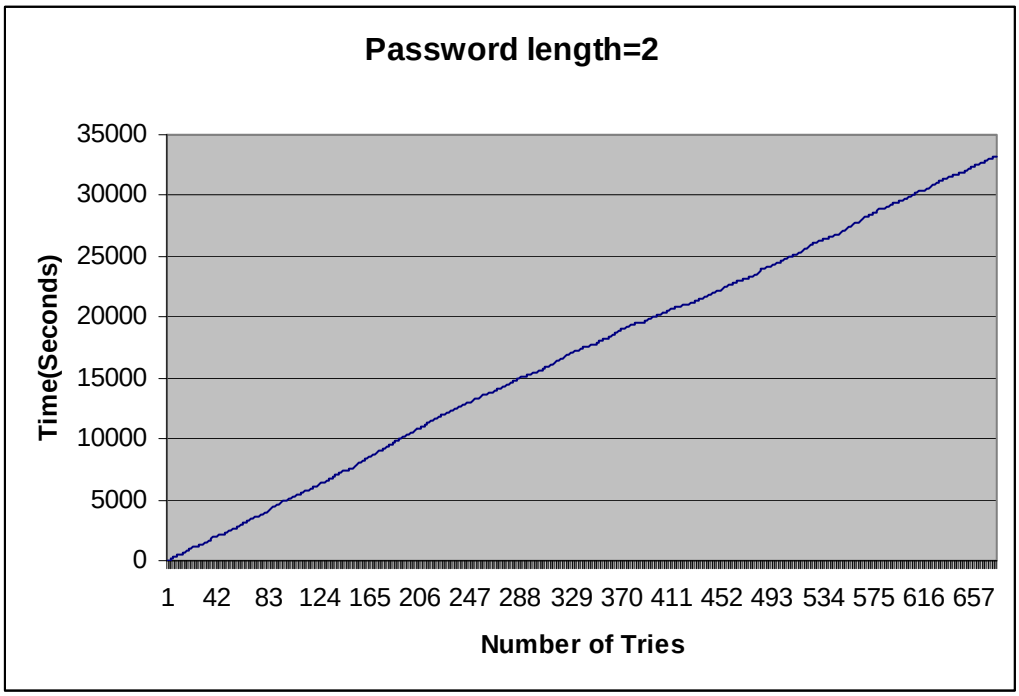

Figure 4.6: Brute Force Attack Findings for password length 2 (alphabet only)

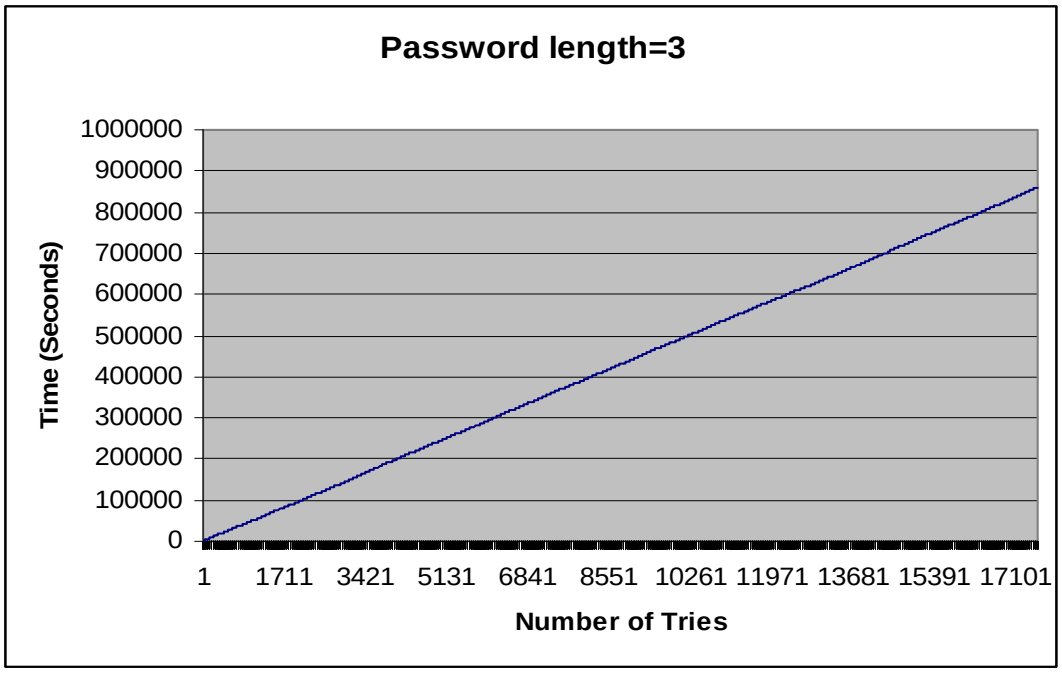

Figure 4.7: Brute Force Attack Findings for password length 3 (alphabet only)

(iii) Insufficient Authorization Attack

We wrote our code in ASP.NET and C# based on the following deception algorithm for Insufficient Authorization Attack.

```
while (true) //this is a service, so it will run forever
do
        group=user_groupdetect(ip_address, user_ID); //detect the user's functional group

        //check if the user has the privilege to perform the operation
        if requested_operation(request,group)="allowed"
                connectionstring=original_database_connection_string
        else
                connectionstring=false_database_connection_string

                delay_in_seconds=random (10-100)
                pause(delay_in_seconds) //introduce a random delay
        end If


        //connect to database according to intrusion detection decision
        connect_database(connectionstring)
        result=execute_SQL()

        return result
end do
```

This algorithm was run on three sets of 1000 requests. The attacks were embedded on those requests for a probability of 0.1, 0.2 and 0.3 (Probability 0.1 means out of 10 requests one is an attack and so on). From the graphs showing the results, we can see that the time to deceive is directly proportional to the insufficient authorization attack probability. As always, we considered 3 seconds of general service time for any requests from the web service consumer end.

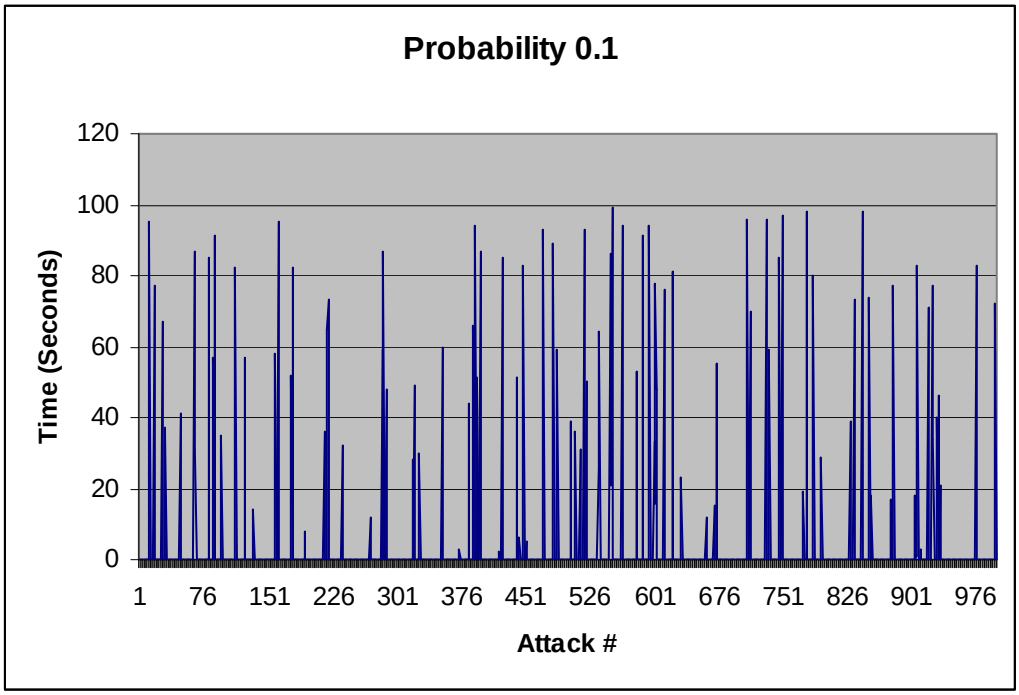

Figure 4.8: Insufficient Authentication Findings for Attack Probability 0.1

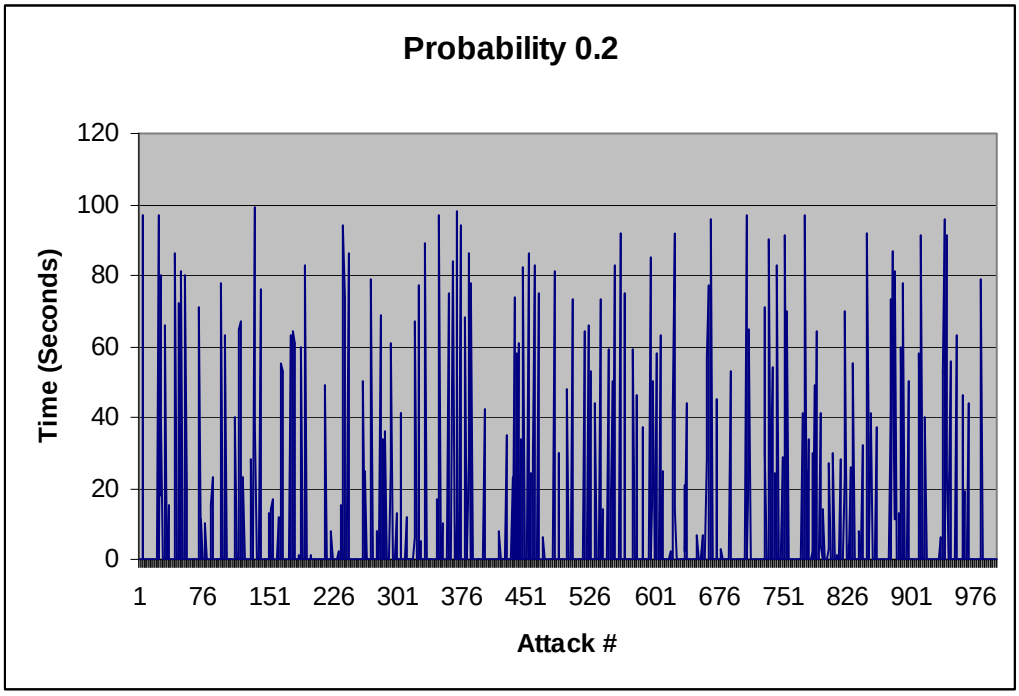

Figure 4.9: Insufficient Authentication Findings for Attack Probability 0.2

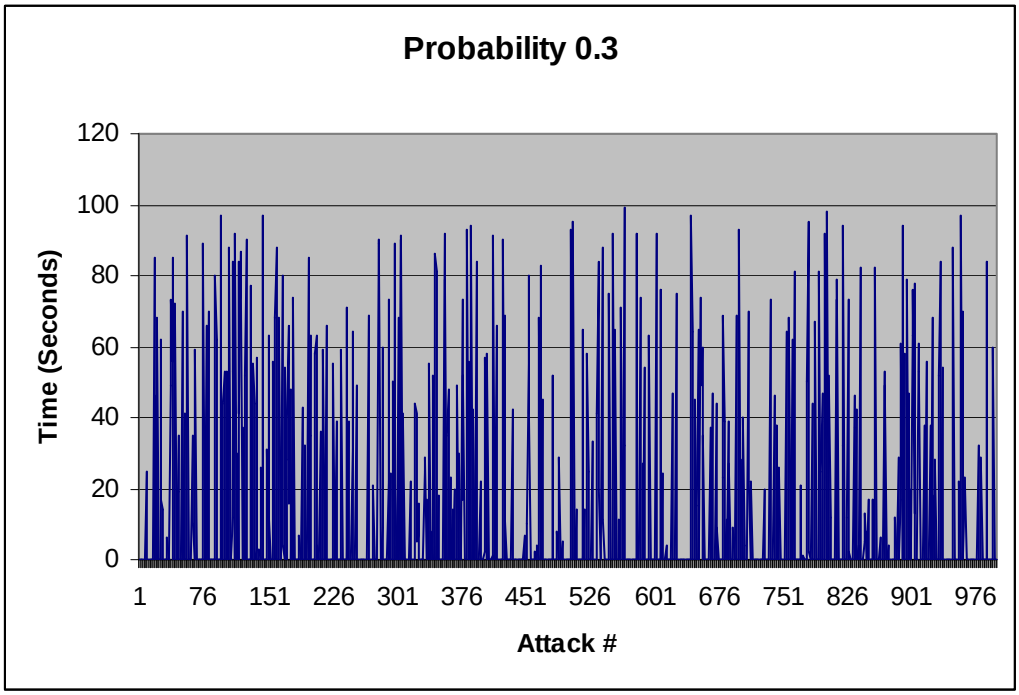

Figure 4.10: Insufficient Authentication Findings for Attack Probability 0.2

4.1 Analysis

Based on our generated simulation data we found that there can be a total of 2880 attacks in a day with an attack probability of 0.1. We considered 3 seconds of general service time spent for any kinds of request made to the server which includes network delay, service discovery time, connecting time to the SOA port, authentication time for a user, detection time for an attack, redirection time to a real database service/ Honeypot. Using deception we reduced the number of attacks to 155. Therefore we reduced 2725 attacks out of 2880 attacks when the attack probability is 0.1; this is a 95% reduction in attacks. Thus we reduced 113 attacks in an hour or 1.8 attacks in a minute.

For attack probability 0.2 we found that only 98 attacks are possible to the web service because of the deception time introduced to the hacker. This saves the system from 5662 attacks in a day, 236 attacks in an hour, and 3.9 attacks in a minute. This is a 98% reduction in attacks. Again for attack probability of 0.3 this system will permit 155 attacks out of 8640 attacks to a system and therefore it will save 8485 attacks a day, 353 attacks an hour and 5.9 attacks in a minute. This is a 98% reduction in attacks.

CHAPTER 5

CONCLUSION

The objective of our work was to use deception for web services in a Service Oriented Architecture as a means of reducing attacks. Our results show that deception is a successful technique to secure web services. In our work we proposed deception to work on top of Web Services. We selected three kinds of attacks and implemented those in our work. We showed that the proposed technique we developed would significantly reduce the percentage of attack to the web services. Our approach can kill the percentage of time an attacker spends in attacking, therefore proportionately the time the attacker is involved in attack a service would be less when compared to a scheme where deception is not employed. Moreover, this technique would also help to eventually outrace back the attacker by deceiving him and collecting information from him.

This work can be extended to the implementation of all the attacks described above. We have only implemented three kinds of attack models in this work.

In addition to this there is a scope to compare the performance of this system to the typical intrusion detection and prevention technique.

In this work we limited our scope within the boundary of a local area network whereas for web services there is no organizational boundary and hence this system should be deployed on the internet for measuring the performance of the system.

In our prototype we only implemented some ways to prevent a particular attack but in the real world attack can be widely diversified and in this scenario our system may not provide very good performance; therefore this implementation can be expanded to prevent large scale attacks.

An attacker can be redirected to several places and future work will also investigate collecting relevant attacker information such as attacker identity, location etc.

REFERENCES

[1] IBM Systems Journal | Vol. 44, No. 4, 2005 - Service-Oriented Architecture http://www.research.ibm.com/journal/sj44-4.html

[2] Mokube, I. and Adams, M. , "Honeypots: concepts, approaches, and challenges", Proceedings of the 45th annual southeast regional conference, Pages: 321 – 326, 2007

[3] Kevin Borders, Laura Falk, and Atul Prakash, "OpenFire: Using Deception to Reduce Network Attacks", Proceedings of the 3rd International Conference on Security and Privacy in Communication Networks, 2007

[4] Karthikeyan Bhargavan, C´edric Fournet, Andrew D. Gordon and Greg O'Shea. "An advisor for web services security policies", Proceedings of the 2005 workshop on Secure web services, Pages: 1 – 9, 2005

[5] Mohammad Ashiqur Rahaman, Andreas Schaad, Maarten Rits, "Towards secure SOAP message exchange in a SOA", Proceedings of the 3rd ACM workshop on Secure web services, Pages: 77 – 84, 2006

[6] Takeshi Imamura, Michiaki Tatsubori, Yuichi Nakamura, “Web services security configuration in a service-oriented architecture”, Special interest tracks and posters of the 14th international conference on World Wide Web, Pages: 1120 – 1121, 2005

[7] Liang Fang, Samuel Meder, Olivier Chevassut, Frank Siebenlist, “Secure password-based authenticated key exchange for web services”, Proceedings of the 2004 workshop on Secure web service, Pages: 9 – 15, 2004

[8] Adams, C. and Boeyen, S., “UDDI and WSDL extensions for Web service: a security framework”, Proceedings of the 2002 ACM workshop on XML security, Pages: 30 – 35, 2002

[9] Angelos D. Keromytis, Janak Parekh, Philip N. Gross, Gail Kaiser, “A holistic approach to service survivability”, Proceedings of the 2003 ACM workshop on Survivable and self-regenerative systems, Pages: 11 - 22 , 2003

[10] Mahantesh Hosamani, Harish Narayanappa, Hridesh Rajan, “Monitoring the monitor: an approach towards trustworthiness in service oriented architecture”, 2nd international workshop on Service oriented software engineering, Pages: 42 - 46 , 2007

[11] Massimo Mecella, Mourad Ouzzani, Federica Paci, Elisa Bertino, “Access control enforcement for conversation-based web services”, Proceedings of the 15th international conference on World Wide Web, Pages: 257 – 266, 2006

[12] Liam O'Brien, Len Bass, Paulo Merson, “Quality Attributes for Service-Oriented Architectures”, Proceedings of the International Workshop on Systems Development in SOA Environments, Pages: 3-3, 2007

[13] Boualem Benatallah, Mohand-Said Hacid, Alain Leger, Christophe Rey, Farouk Toumani, “On automating Web services discovery”, The International Journal on Very Large Data Bases archive, Volume 14 , Issue 1. Pages: 84 – 96, 2005

[14] Jiehan Zhou, Daniel Pakkala, Juho Perälä, Eila Niemelä, “Dependency-aware Service Oriented Architecture and Service Composition”, Proceedings of the IEEE International Conference on Web Services, Pages: 1146-1149, 2007

[15] Al-Masri, E., Mahmoud, Q.H, “Discovering the best web service”, Proceedings of the 16th international conference on World Wide Web, Pages: 1257 – 1258, 2007

[16] Ivan J. Jureta, St´ephane Faulkner, Youssef Achbany, Marco Saerens, “Dynamic Web Service Composition within a Service-Oriented Architecture”, Proceedings of the IEEE International Conference on Web Services, Pages: 304-311, 2007

[17] Pornpong Rompothong, Twittie Senivongse, “A query federation of UDDI registries”, Proceedings of the 1st international symposium on Information and communication technologies, Pages: 561 – 566, 2003

[18] Eyhab Al-Masri and Qusay H. Mahmoud, “Crawling multiple UDDI business registries”, Proceedings of 16th International World Wide Web Conference, Pages: 1255-1256, 2007

[19] Manoj Paul, S. K. Ghosh, “An approach for service oriented discovery and retrieval of spatial data”, Proceedings of the international workshop on Service-oriented software engineering, Pages: 88 – 94, 2006

[20] Zhou Zhu and James Bailey, “Fast Discovery of Interesting Collections of Web Services”, Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence, Pages: 152-160, 2006

[21] Yujian Fu, Zhijiang Dong, Xudong He, “Modeling, validating and automating composition of web services”, Proceedings of the 6th international conference on Web engineering, Pages: 217 – 224, 2006

[22] Luciano Baresi, Carlo Ghezzi, Antonio Miele, Matteo Miraz, Andrea Naggi, Filippo Pacifici, “Hybrid service-oriented architectures: a case-study in the automotive domain”, Proceedings of the 5th international workshop on Software engineering and middleware, Pages: 62 – 68, 2005

[23] Kerry Taylor, Tim Austin, Mark Cameron, “Charging for information services in Service-Oriented Architectures”, Proceedings of the IEEE EEE05 international workshop on Business services networks, Pages: 16 – 16, 2005

[24] G. Denaro, M. Pezzé, D. Tosi, Daniela Schilling, “Towards self-adaptive service-oriented architectures”, Proceedings of the workshop on Testing, analysis, and verification of web services and applications, Pages: 10 – 16, 2006

[25] Patrick, Paul, “Impact of SOA on enterprise information architectures”, Proceedings of the ACM SIGMOD international conference on Management of data, Pages: 844 – 848, 2005

[26] Jia Zhang, Jen-Yao Chung, Carl K. Chang, “Migration to web services oriented architecture: a case study”, Proceedings of the ACM symposium on Applied computing, Pages: 1624 – 1628, 2004

[27] Yves Petinot, C. Lee Giles, Vivek Bhatnagar, Pradeep B. Teregowda, “A service-oriented architecture for digital libraries”, Proceedings of the 2nd international conference on Service oriented computing, Pages: 263 - 268, 2004

[28] K. Bhargavan, C. Fournet, A.D. Gordon, “A semantics for web services authentication”, Journal of Theoretical Computer Science, Volume 340 , Issue 1, Pages: 102 – 153, 2005

[29] Frank Kargl, Joern Maier, Michael Weber, « Protecting web servers from distributed denial of service attacks", Proceedings of the 10th international conference on World Wide Web, Pages: 514 – 524, 2001

[30] Carlos Gutiérrez, Eduardo Fernández-Medina, Mario Piattini, "Web services enterprise security architecture: a case study", Proceedings of the workshop on Secure web services, Pages: 10 – 19, 2005

[31] Andrew D. Gordon, Riccardo Pucella, "Validating a Web service security abstraction by typing", Proceedings of the ACM workshop on XML security, Pages: 18 – 29, 2002

[32] SECURITY ARCHITECTURES Service Oriented Security Architecture, http://www.arctecgroup.net/ISB1009GP.pdf

[33] To Trap A Thief, http://www.computerworld.com/networkingtopics/networking/lanwan/story/0,10801,59072,00.html, Modified: 04-02-2001

[34] Threat Classification - Web Application Security Consortiumhttp://www.webappsec.org/projects/threat/classes_of_attack.shtml, 2008

APPENDIX

The Web service client was configured by the following files to connect to the server. In Microsoft Visual Studio there are options to bind a web service consumer with a service provider by service discovery file.

```
<?xml version="1.0" encoding="utf-8"?>
<discovery xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.xmlsoap.org/disco/">
  <contractRef ref="http://deepak-pc/WebService.asmx?wsdl"
docRef="http://deepak-pc/WebService.asmx"
xmlns="http://schemas.xmlsoap.org/disco/scl/" />
  <soap address="http://deepak-pc/WebService.asmx"
xmlns:q1="SQLInjction" binding="q1:SQLInjSoap"
xmlns="http://schemas.xmlsoap.org/disco/soap/" />
  <soap address="http://deepak-pc/WebService.asmx"
xmlns:q2="SQLInjction" binding="q2:SQLInjSoap12"
xmlns="http://schemas.xmlsoap.org/disco/soap/" />
</discovery>
```

Figure A.1: The Discovery File for Web Service Consumer

Instead of UDDI we used service discovery files to directly map the server to the client. Implementing UDDI requires a strong tool like IBM Websphere to implement. However, Microsoft Visual Studio generates discovery map file if we use the visual tools to map a server with a client.

```
<?xml version="1.0" encoding="utf-8"?>
<DiscoveryClientResultsFile
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Results>
    <DiscoveryClientResult
referenceType="System.Web.Services.Discovery.DiscoveryDocumentReferen
ce" url="http://deepak-pc/WebService.asmx?disco"
filename="WebService.disco" />
    <DiscoveryClientResult
referenceType="System.Web.Services.Discovery.ContractReference"
url="http://deepak-pc/WebService.asmx?wsdl"
filename="WebService.wsdl" />
  </Results>
</DiscoveryClientResultsFile>
```

Figure A.2: The Discovery Map File for Web Service Consumer

Microsoft Visual Studio generates WSDL files for only service consumers and not for the servers. Instead of writing WSDL files, GUI tools can be used to generate this WSDL file.

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:tns="SQLInjction" xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
targetNamespace="SQLInjction"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <s:schema elementFormDefault="qualified"
targetNamespace="SQLInjction">
      <s:element name="VerifyLogin">
```

```
            <s:element minOccurs="0" maxOccurs="1" name="user"
type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="password"
type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="VerifyLoginResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1"
name="VerifyLoginResult" type="s:int" />
          </s:sequence>
        </s:complexType>
      </s:element>
    </s:schema>
  </wsdl:types>
  <wsdl:message name="VerifyLoginSoapIn">
    <wsdl:part name="parameters" element="tns:VerifyLogin" />
  </wsdl:message>
  <wsdl:message name="VerifyLoginSoapOut">
    <wsdl:part name="parameters" element="tns:VerifyLoginResponse"
/>
  </wsdl:message>
  <wsdl:portType name="SQLInjSoap">
    <wsdl:operation name="VerifyLogin">
      <wsdl:input message="tns:VerifyLoginSoapIn" />
      <wsdl:output message="tns:VerifyLoginSoapOut" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="SQLInjSoap" type="tns:SQLInjSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
/>
    <wsdl:operation name="VerifyLogin">
      <soap:operation soapAction="SQLInjction/VerifyLogin"
style="document" />
      <wsdl:input>
        <soap:body use="literal" />

</wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="SQLInjSoap12" type="tns:SQLInjSoap">
    <soap12:binding
transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="VerifyLogin">
      <soap12:operation soapAction="SQLInjction/VerifyLogin"
style="document" />
```

Figure A.3: The WSDL File for Web Service Consumer

VITA

SHAH MD. EMRUL ISLAM

Candidate for the degree of

Master of Science

Thesis: AN APPROACH TO IMPLEMENT SECURITY IN SERVICE ORIENTED ARCHITECTURE USING DECEPTION TECHNIQUE

Major Field: Computer Science

Biographical:

Personal Data: Born in Rangpur, Bangladesh, On January 7, 1979, the youngest son of Shah Fatah and Hosneara Begum.

Education: Graduated Higher Secondary Certification from Carmichael College, Rangpur. Received Bachelor of Science degree in Computer Science & Engineering from Ahsanullah University of Science and Technology, Bangladesh in November 2002. Received Master of Science degree in Information and Communication Technology from Bangladesh University of Engineering and Technology, Bangladesh in October 2004.

Professional: Currently working for Oklahoma Cooperative Extension Services as a Graduate Research Assistant (Database Manager), Dept. of Agriculture Economics, Oklahoma State University, Stillwater. Previously worked for Hewlett Packard Malaysia, Dhaka Bank Limited Bangladesh, United Nations Development Programme, Bangladesh. Mr. Islam is an IBM Certified Specialist for AIX and HP Certified System Administrator for HP-UX.

Name: Shah Md. Emrul Islam Date of Degree: May, 2008

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: AN APPROACH TO IMPLEMENT SECURITY IN SERVICE ORIENTED ARCHITECTURE USING DECEPTION TECHNIQUE

Pages in Study: 55 Candidate for the Degree of Master of Science

Major Field: Computer Science

Scope and Method of Study:

Service Oriented Architecture is a relatively new field in computing. However, web services in a Service Oriented Architecture are usually open and vulnerable to attacks. Intrusion detection is a technique widely used for protecting web services. In this thesis, we use deception on top of Intrusion Detection in a Service Oriented Architecture. We implemented three attacks on web services, namely, SQL Injection attack, Brute Force attack and Insufficient Authorization attack. We developed algorithms to deceive against these attacks. From our result we saw that deception wastes the time and resources of the attacker and furthermore is able to reduce attacks by more than 90 percent. Deception is therefore complementary to intrusion detection and can be effectively used to protect web services.

ADVISOR'S APPROVAL: Johnson Thomas .