University of Piraeus
Department of Digital Systems
Postgraduate Program in Digital Systems Security

Master Thesis

Security of Web Service-based Applications

Evangelos Markopoulos

Supervisor_ Costas Lambrinoudakis

June 2012

Table of Contents

List of Figures

List of Tables

"Security is not a number of features, but a system process"

1.1 The purpose of this Research Thesis

The main purpose of this Research Thesis is to present and analyze security considerations of Web Services Technology. The writing of the Thesis is kept in a simple and friendly pace, thus enabling the target audience to get good understanding of the following topics:

- What is Web Services Technology and how it works.
- How easy is the development of a simple Web Service.
- What are the security considerations of Web Services and how security is assessed.
- Implementation of security mechanisms concerning the Web Service itself.

Penetration testing tools and processes are fairly presented as a good entry point to the Security Assessment Industry.

The content of the Research Thesis is addressed to the following Target audience:
(a) Developers with limited experience in Web Services and especially in Web Services Security will learn the importance of securing the service from the code (b) Newcomers in penetration testing will gain basic knowledge of tools and processes used for testing Web Applications and Web Services Security (c) Students of security-related faculties will be encouraged for further research in Web Services Security.

1.2 Challenges

The first challenge before starting writing this Thesis was to understand thoroughly the Web Service Technology and set the desirable outcomes for this Thesis to present. Next, it came the major challenge of designing and implementing a functional, operational but most of all public Web Service, combining knowledge from multiple sectors such as software development process (SDLC) and Web Technologies. This service underlies the running example of the whole Thesis. It has been tried to represent a real/working service on the web, highly consumed by users worldwide. Considering the limited programming experience of the writer, it was a huge challenge to create and implement a Web Service from scratch. Another great challenge was to assess the security of the Web Service and then integrate security features in its initial implementation.

Another major challenge constitutes the security itself discussing and analyzing threats, vulnerabilities and countermeasures based mostly on the main Web Service example. It has been put a lot of effort by the writer to keep a simple, well-organized structure of the Thesis through the analysis flow, trying to connect theory and findings with the working example, thus providing readers with better understanding.

1.3 Conventions

The first assumption made is the attack scenario. I assume I do not know anything about the service that I am going to test and therefore I conduct a Blackhat (or Blackbox) penetration testing. The penetration testing process is further analyzed in PART III under the penetration testing section.

Another assumption made is about the creation of credentials for end users. Users have to follow -a priori- a typical registration process in order to acquire the appropriate credentials.

The third assumption is made on service consumption. With the existing mechanism, the client application has been developed (UI) is always authenticated to the database through a static, private connection string. As a consequence, when the service is going to be consumed by a third party (for example an external java client) will have to be provided either the same credentials with the client application or change the process connecting to the database.

Finally, all examples considering the architecture of Web Services, for example SOAP and WSDL use representations of my own Wcf Web Service implantation which is presented and analyzed in PART II.

1.4 Structure of this Research Thesis

To facilitate the reader, this section contains a brief description of the remaining chapters of this research thesis. Along with the description, the main points of each chapter are summarized and presented.

This research thesis consists of three parts.

PART I, is concerned with the analysis of Web Services. It starts with a basic understanding on what Web Services mean, where they derive from and for what reasons Web Services have become so successful. Then, it presents the Web Services architecture. What are the technologies behind and how they work? Who decides about the fate – development of Web Services? Understanding the key components of Web Services is a very important pre-requisite for next Part.

PART II, goes over the design and development of a simple Web Service based on Microsoft's Windows Communication Framework. It is presented the implementation environment as well as analysis of the design of the Wcf Web Service. The scope is to present the ease and simplicity in designing and implementing a new Web Service from scratch.

PART III, consists the bigger and most important part of this thesis concerning the security of Web Service. It starts referring to the bibliography and theory of security specifications and

mechanisms about Web Services. In order to facilitate the reader, a layer approach is adopted, examining Transport, Message and Application Security focusing, primarily, on the Message Layer Security. Then, direct and indirect to Web Services attacks are presented, divided into XML-based attacks and Web Application Attacks. Next, follows a section on penetration testing. Penetration testing is based on a hypothetical attack scenario for the implemented Wcf Web Service described in Part II. After the vulnerability assessment of Web Services and the final results, security countermeasures for the Wcf Web Service are proposed and implemented. As a result, a new security-revised version of the implemented Wcf Wen Service is provided.

At the end, there are conclusions and some future considerations.

1.5 Limitations of this Research Thesis

a. Service consumption. The Web Service and its client application have been developed under ASP.NET Microsoft technology. It has not been attempted to create any client using other technologies (for instance java) in order to better show interoperability of the Web Service.
b. Limited functionality. There are only two operations described in the service. Although extensibility is possible, any further functionality may have technical implications especially concerning the security of the service.
c. Penetration Testing. Penetration testing is presented in this thesis in a more general and abstract way. Although, the basic steps existing in every penetration testing methodology – model are presented, the whole testing process in not fully organized and documented according to a certain model. Furthermore, penetration testing tools and techniques are not described – explained in detail but rather stay as simple as possible in order to illustrate some of the possible threats – attacks.

PART I WEB SERVICES ANALYSIS

Understanding Web Services

Web Services technology allows applications to communicate with each other regardless of platform and programming language. A web service is a software interface which describes a collection of functions that can be accessed from the network via standard XML messages. It uses standards based on XML language to describe a function (operation) for execution and data to be exchanged with another application. A group of web services that interact with each other defines a web services application.

A primary goal of Web Services was to unlock a new generation of e-commerce applications through the machine-to-machine communication over the internet achieving perpetual interoperability among systems/technologies.

In terms of describing the use of computer systems, Web Services introduced a new business perspective. Moving from the old model where Business = Existing Infrastructure [Applications] + Business Rules, the addition of Web Services to the equation results in a new situation where Existing Infrastructure [Applications] + Business Rules + Web Services = New Business. The outcome is that Web Services open up old functionality creating new revenues for business.

I will start analyzing Web Services by examining the architecture upon which Web Services base their logic. This is called Service Oriented Architecture or SOA. Next, it is going to be provided enough information about defining Web Services and highlighting their benefits. Further analysis will be provided about classes and styles of Web Services as well as their categories. The most important information concerns the technologies of Web Services addressing the way they work, thus enabling the reader to establish a solid knowledge about Web Services Stack.

2.1 Service Oriented Architecture

Service-oriented architecture (SOA) is a conceptual architecture that uses services to support business processes. The concept of service is the cornerstone of SOAs through which it is injected a new orientation of the business logic. Instead of changing a business process made by several services, SOA introduces innovation by inserting one or more new services in that process. This is a new notion of modifying a business process by adding, removing or combining old and new services together in order to produce a whole new process without writing or modifying any piece of custom code. Therefore SOA presents an easier, faster and less expensive way of doing business. According to SAP, its approach to SOA is “a blueprint for adaptable, flexible and open IT infrastructure for developing service-based, enterprise-scale business solutions” [41].
SOA enables organizations to utilize their IT infrastructure and align it with business requirements. IT systems can be more easily integrated automating business processes reducing at the same time the cost of acquiring and maintaining IT systems and applications.
Especially for applications and business processes development, SOA provides rapid and easy reusability of elements (services).

In our rapidly changing world, a major concern is how to enable information sharing across departments - functions and between enterprises – industries, automating business processes. Organizations had always difficulty to ensure interoperability of hundreds of systems/applications running either departmental, enterprise wide or inter-enterprise. Even more difficult is running projects of updating or changing all or some of these operational systems. But the most difficult part was to ensure interoperability across their domain with their customers and partners. The Internet became the most widespread and important way of communication among enterprises, their business partners and customers where everything now connects to the Web.

Web services are an ideal platform for SOA. They are the modern answer to application integration through SOA architecture. Their difference from other SOA services is that they mostly live on the Web enabling communication across the Internet. The following figure shows the SOA model. As we will notice in the following section, SOA model is actually the Web Services programming model or Web Service Architecture. Therefore, either we refer to SOA services or Web Services it is actually the same thing.

Figure 1 represents the SOA theoretical model for Web Services

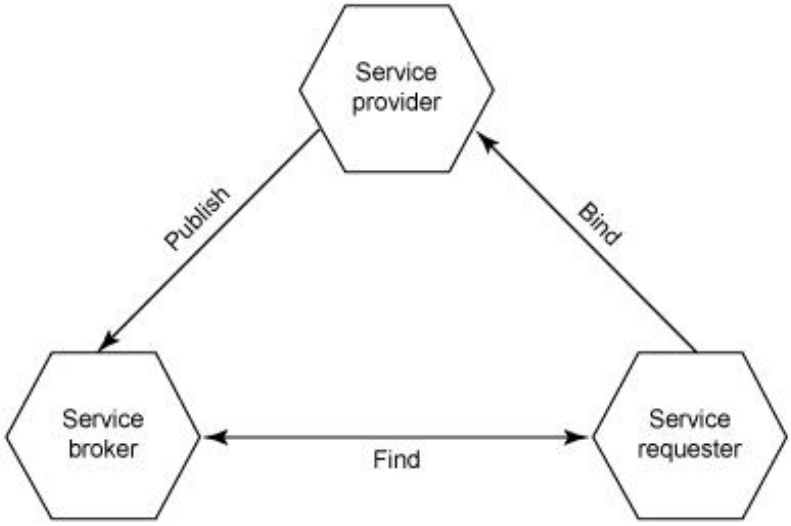


Figure 1: SOA model

Service Provider: It implements the service and makes it available on the Internet.

Service Requester: It can be considered any consumer of the service. It is usually an application which tries to find information about the operations of the service as well as its location in order to consume it.

Service Broker: It is also called service registry. This is a well-known application, a centralized registry of services where developers can publish new services or find existing ones. It therefore serves as a centralized clearinghouse for companies and their services.

2.2 Defining Web Services

Web Services represent a sub-domain of the service science similar to IT services (or e-services). Actually Web Services belong to the IT domain with the difference there are basically web oriented1. However, Web Services do not do not necessarily follow the same design principles of the Web neither rely on the same core formats and protocols.

There is no prevalent, universally accepted definition about what Web Services is. On one hand, this fact denotes the open, global nature of Web Service technology as a combination of elements across computer science. On the other hand, multiple organizations, vendors, communities and academia have tried to define Web Services on their perspective.

To begin with, the World Wide Web Consortium (W3C) defines a Web service as "a software system designed to support interoperable machine-to-machine interaction over a network" [WS-Arch]. In a more detailed description, W3C2 describes Web Services as: "software applications identified by a URI, whose interface and bindings are capable of being identified, described and discovered by XML artifacts and support direct interactions with other software applications using XML based messages via Internet-based protocols"[13].
Preist[3] defines Web services as computational entities that can be assembled and accessible over the Internet using Web service standards and protocols performing functions or executing business processes.
In the J2EE 1.4 environment “A Web service is a software application that conforms to the Web Service Interoperability Organization's Basic Profile 1.0”and can be accessed remotely using different XML-based languages.

Figure 2, illustrates the place of Web Services inside the Cloud Service Stack

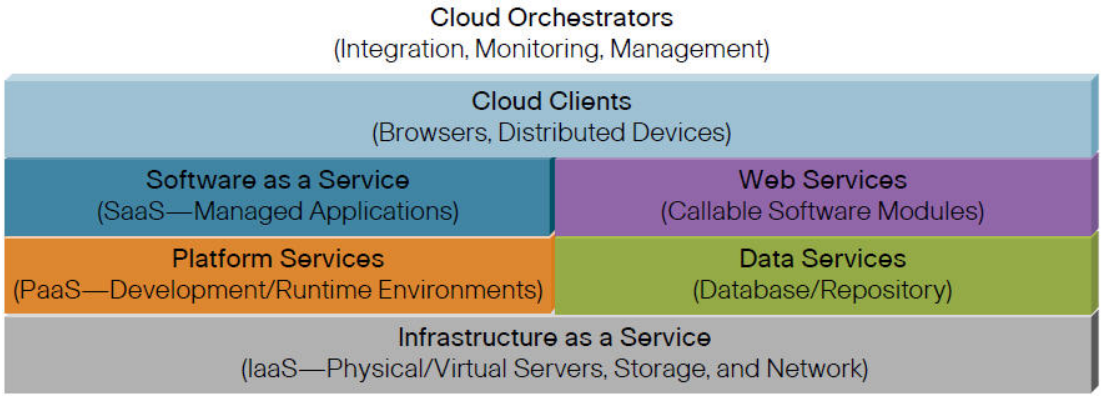


Figure 2: Cloud Service Stack Source: Cisco IBSG, 2008

[1] Web Services apart from the Internet can be applied in any Intranet environment

[2] http://www.w3.org/TR/ws-desc-reqs/

[3] Preist, C.: A conceptual architecture for semantic web services. In: Proceedings of the International Semantic Web Conference 2004 (ISWC 2004) (2004)

In my perspective, what I embrace most is the notion of Web Services as programmable application components accessible via standard Web protocols which enable machine-to-machine interaction. This is an application-centric Web model that actually Web Services introduce. Trying to recapitalize and provide a more clear definition:

"*Web Services are a technology that allows applications to communicate regardless of platform and programming language. A web service is a software interface (software interface) that describes a collection of functions that can be accessed from the network via standard messages XML. It uses standards based on XML language to describe a function (operation) for execution and data to be exchanged with another application. A group of web services that interact with each other determines an application web services*".

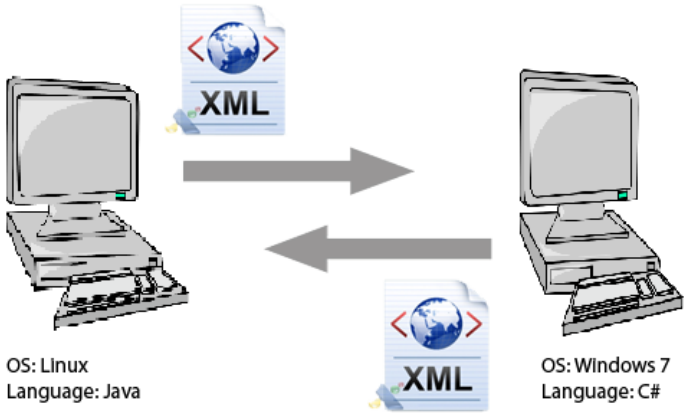


Figure 3: A basic Web Service

In the next paragraphs, they are going to be presented some basic characteristics of Web Services. These characteristics are at the same time the success factors for the adoption of Web Services compared to the forerunner technologies.

2.2.1 Success factors of Web Services

Web Service Technology offers numerous advantages intimately related to its Service Oriented Architecture.

A key benefit of the adoption of Web services technology is their ability to deliver integrated, interoperable solutions between applications and systems. Actually, the root problem which led to the need of Web Services was Enterprise Application Integration. I like to use the following quote: "All for business and business for business". Web Services, because of their nature, can interconnect applications and systems not only within organizations and intranets but also across enterprise boundaries, the Internet. Their XML-based nature allows them to focus on the data to be shared among services or applications whatever the execution environment is. Web Services are platform and language independent thus not needing for a specific development environment. Web services ability

to work with any type of implementation software enhances the IT value and reduces costs, especially labor costs.

Another characteristic of Web services is reusability. The same service can serve many other applications again and again. This is of high importance in service orientation in order services to get their highest value. Web Services provide, also, efficiency by enabling quick and easy development where IT stuff performs the bulk of their activities. Moreover, enable IT people to focus on their technical issues while business people to concentrate on how to run business regardless the existing technology (removes technicality). Another important benefit is loose technology coupling. Web Services can be executed independently and communicate with any other service or type of implementation software. The removed complexity of Web Services as well the loosely coupled, reusable software components, led to an easy and rapid, on-the-fly, software development especially beneficial for B2B integration but also B2C e-Commerce. Web Services characteristics such as flexibility, dynamism, adaptability, scalability and maintainability; enable business to improve their Performance (high throughput, timely results) and Statelessness[1] of Web Service implementations.

Finally, Web Services SaaS initiatives signify their open-source orientation which has as a consequence the active complicity of developer community, non-profit organizations and agencies but the whole IT industry in general, which denotes the persistence and evolution of this technology.

2.2.2 Disadvantages of Web Services

1. Lack of standards. The existence of various standards about Web Service Technologies may cause failures[2] or incomplete transactions. This can due to incompatibilities between the versions of SOAP, UDDI, HTTP, and WSDL that are used between the two sides of the communication or due to the implementation of vendor-specific solutions.

2. Availability. In the Internet era, 100% uptime a web site - web server is the biggest concern depending on multiple factors.

3. Guaranteed execution and performance issues. Because of the use of HTTP, it is not always possible to guarantee delivery of a request / response. On the other hand, HTTP is very transactional and although this is enabling a server to handle many clients it is not possible to maintain a long-term connection with those clients. As a consequence, there is spending much time in creating and terminating connections with clients. This is particularly painful in cases clients need to perform large number of calls. Moreover, much time is

[1] Statelessness is a valuable asset to their availability and ability to accommodate dynamic workloads

[2] Web Services lack a transaction model to allow for a rollback in case of failures

needed in translating XML[1] messages from received / send to the clients in a SOAP envelop. All the above create performance issues.

4. Immutable interfaces. Any changes of the methods provided or the parameters of the service can cause client applications to crash. In this case it is not always possible to know all of the clients that are using the service and inform them for the changes. As a result, changes on the existing code consists a very complicated process.

5. Matching requirements. Web Services are one for all solution in a sense that it is not possible for a specific service to meet each special requirement of every customer.
6. Granularity. Web Services are often too granular to be efficient in everyday business operations especially for demanding business processes to work.

2.2.3 Web Services Examples

Since the Governing Bodies of Web Service Technology set up a set of standards, most vendors adhered at a minimum in some of these specifications (for instance conformance to XML, SOAP specifications). As we can see, the adoption of Web Services is across industries or application type. The biggest industry of adopting Web Services was from the beginning the IT industry. E-procurement was maybe the first and most directly affected industry which through new forms of B2B, B2C (e-commerce) and B2G web service-based procedures promoted business services and electronic data interchange.

To continue, package delivery service industry, electronic payments processing, banking and financial services as well as retail merchandise were next in the line of early adopting Web Services. Through the years, Web Services invaded everywhere from transportations, telecommunications and media to manufacturing, healthcare, education and even public sector. E-government is a heavy example of adopting web service technology, collecting all the benefits that this technology offers.

There are numerous well-known examples of international brand names which have successfully implemented Web Services. Microsoft's Passport for authentication services is an early example of Web Service technology adoption. Google, Yahoo and EBay are between the best examples that provide Web Services implementations for everyday use by millions of people. American Airline's pioneering automated reservation system (SABRE), Dell's on-demand manufacturing supply chain management system and Amazon's Web Services are some other very notorious examples. Other classic examples include FedEx and DHL in package delivery services, TUI Europe's leading travel group and Wal-Mart's inventory management system where Web Services are used as Best Supply Chain Management Practices. The list continues and it is endless. Even ERP vendors such as SAP utilize Web Services although in a more Enterprise Oriented Architecture. Nowadays, there is almost none multinational company or big web site/portal that do not use or intent to use Web Services.

[1] XML tends to take more bytes to encode data than the equivalent binary representation.

2.3 Web Services Architecture

The Web Service Architecture can be fairly described by the SOA model as illustrated in previous section. However, the following Figure (5) illustrates Web Service Architecture including all its major components that will be presented in this section.

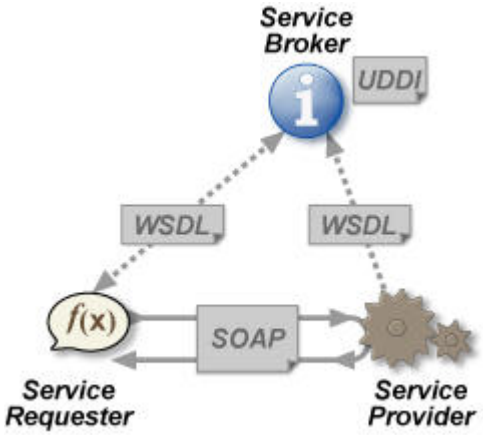


Figure 4: Web Services Model (Source: Wikipedia)

Illustration of Web Service Protocol stack will provide the reader with a more clear understanding of the Web Service Architecture. Therefore, the basic layers of this protocol stack can be presented as it follows:

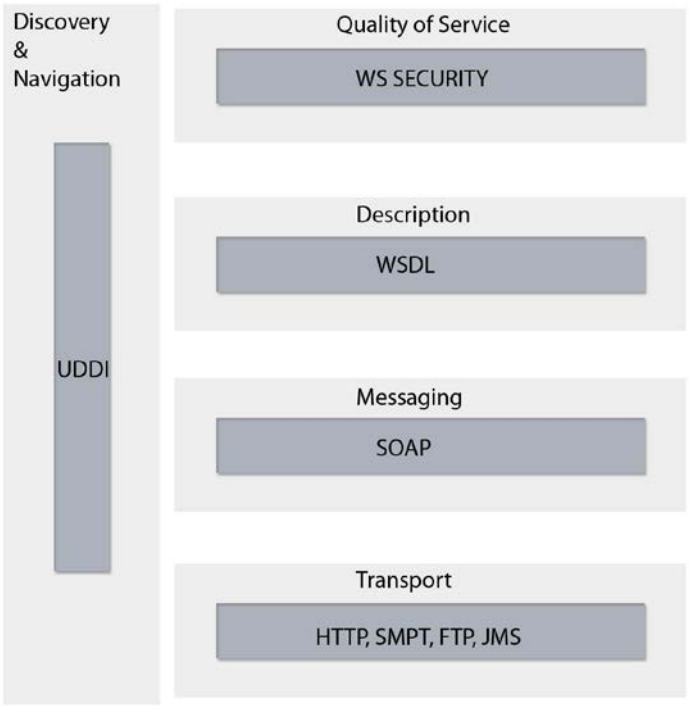


Figure 5: Web Service Protocol Stack

Service Transport: It is responsible for transporting messages between applications. It includes well known transport protocols such as HTTP/HTTPS, SMTP, FTP.

XML Messaging: This is the data layer where messages are encoded in a common XML format such as SOAP or XML-RPC in order to be understood at either end.

Service Description: It is responsible for describing the public interface to a specific Web Service. It is done through the use of WS-Policy and XSD but mainly by the WSDL.

Service Discovery: Provides "publish" and "find" services through a centralized registry. UDDI is responsible for this layer.

Quality of Service: Quality of Service (QoS) refers to non-functional properties of services which can be measured based on different parameters. Depending upon the type of service, different quality measures are taken into consideration. The most common ones, usually considered for e-Services, are the following: Availability, Accessibility, Integrity, Performance (the number of service requests served at a given time period), Reliability, regulatory (according to a set of rules/standards, security [1].

2.3.1 Core Pillar Technologies

This section describes the Key Components or Key Specifications of Web Services.

Simple Object Access Protocol (SOAP)

This is the messaging layer in the Web Services Stack.

SOAP is XML. In an effort to provide a kind of definition, SOAP is an extensible XML messaging protocol for machine-to-machine information exchange[1] which is considered as the foundation for Web Services. SOAP provides a simple and consistent one-way transmission mechanism from a SOAP sender to a SOAP receiver, where any application can participate in an exchange as either sender or receiver. The XML nature of SOAP provides it with platform and language independent characteristics thus making it a widely adopted specification. SOAP messages may be combined to support many communication behaviors, including request/response, solicit response, one-way asynchronous messaging, or event notification. SOAP is a high-level protocol that defines only the message structure and a few rules for message processing. It is completely independent of the underlying transport protocol, so SOAP messages can be exchanged over HTTP, SMTP, FTP or JMS transport protocols. However, HTTP is the most frequently used transport protocol for SOAP messages.
Next, there is a short description of SOAP specification and analysis of the structure of a typical SOAP message.

[1] Exchange of services and data

Three core parts:

A) SOAP envelope. It defines rules for encapsulating data being transferred between computers. On one hand, this data can include information about the method name to invoke, method parameters or return values. On the other hand, data can include information about processing the envelope contents or encoding error messages.

B) Data encoding rules. SOAP includes its own set of conventions for encoding data types in order for data to be exchanged between two parts. These conventions are usually based on W3C XML Schema specification [7].

C) RPC conventions. SOAP uses Remote procedure calls and responses conventions for messaging. Usually messages come in pairs as request/response. However, SOAP doesn't require every request to have a response, or vice versa, but it is common to see the request-response pairing.

3 SOAP Message Exchange Patterns (MEPs)

a) one-way

b) request / response

c) peer-to-peer conversations

Example of a simple SOAP request / response:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:tem="http://tempuri.org/">
   <soapenv:Header/>
   <soapenv:Body>
      <tem:GetModelsByPriceRange>
         <!--Optional:-->
         <tem:priceRange>?</tem:priceRange>
      </tem:GetModelsByPriceRange>
   </soapenv:Body>
</soapenv:Envelope>
```

Figure 6: A simple SOAP request from “my Wcf Service” implementation

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:tem="http://tempuri.org/"
xmlns:mar="http://schemas.datacontract.org/2004/07/MarkClassLibrary">
   <soapenv:Header/>
   <soapenv:Body>
      <tem:GetModelsByPriceRangeResponse>
         <!--Optional:-->
         <tem:GetModelsByPriceRangeResult>
            <!--Zero or more repetitions:-->
            <mar:ModelObj>
               <!--Optional:-->
               <mar:ModelDescription>?</mar:ModelDescription>
               <!--Optional:-->
               <mar:ModelID>2</mar:ModelID>
            </mar:ModelObj>
         </tem:GetModelsByPriceRangeResult>
      </tem:GetModelsByPriceRangeResponse>
   </soapenv:Body>
</soapenv:Envelope>
```

Figure 7: A simple SOAP response from “my Wcf Service” implementation

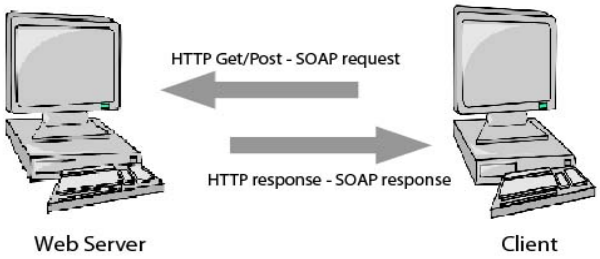


Figure 8: Illustration of SOAP message exchange between an HTTP server and a client

Analysis of the above SOAP messages:

As we can notice from figures 7 - 8, the SOAP message (request and response) contains the following basic elements:

i) An `Envelope` element. This is a mandatory element which identifies the XML document as a SOAP message. Inside it, `Envelope` includes a "sub-element" which is another mandatory element called "`body`" element. `Body` encapsulates the main "payload" of the SOAP message containing call and response information.

ii) A `Header` element. It contains header information. This is an optional element with intentionally open-ended details which offers a flexible framework for specifying additional application-level requirements. It often happens to use `Header` in order to nest security specifications. Common examples could include authentication and confidentiality mechanisms such as XML Digital Signatures or XML Encryption. When `Header` element is included inside a SOAP message, it can itself include three more attributes: the `Actor` or `Role`, `MustUnderstand` and `Relay` attribute. The first one specifies the intended recipient of the SOAP message while the second one indicates where the Header element is optional or mandatory. This denotes that in case of mandatory, the recipient must, at first, understand and then make proper use of the information that the `Header` element contains. The last one indicates whether a SOAP header block targeted at a SOAP receiver, must be relayed if not processed.

To continue, SOAP uses an XML namespace in order to disambiguate XML elements and attributes but also state SOAP version. The default namespace for SOAP envelope is: "`http://schemas.xmlsoap.org/soap/envelope/`," which declares that the message uses SOAP 1.1 version. The namespace `xmlns:tem="http://tempuri.org/` is the test default namespace URI used by Microsoft development products, like Visual Studio and therefore could be replaced by a more meaningful URI. Concerning the SOAP request message, the only element which is included in the `body`, is `GetModelsByPriceRange` which corresponds to the remote method name. On the contrary, on the SOAP response

message body the element that corresponds to the initial request is `GetModelsByPriceRangeResponse.`

The above example includes the most important elements of SOAP message. What it does not include is a `fault` optional element which contains error and status information as well as a namespace for SOAP encoding and data types. SOAP includes a built-in set of rules for encoding data types although these rules are not required. In the above example, the encoding rules are implemented directly by the development kit used for the Web Service and are therefore hidden. If shown, it could be as following:
`soap:encodingStyle=" http://schemas.xmlsoap.org/soap/encoding">`

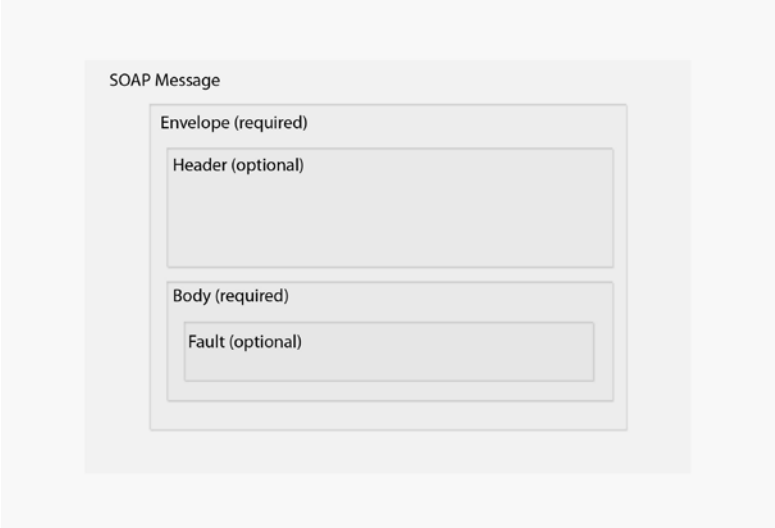


Figure 9: Main elements of the XML SOAP message

Extensible Markup Language (XML)

XML is the most basic technology of Web Services which consists the transmission layer of the Web Service Stack. XML is the base meta-language for Web Services to define communication.

According to OWASP, XML is a standard for exchanging structured data in textual format. The basic elements of an XML document are two: a) the XML Schema and b) the Document Type Definition (DTD) showing both the validity of the document.
There are lots of other standards in the XML family such as XSLT, XSD, XPath, XQuery and XML-Signature.

Similar to HTML, XML consists of elements, attributes and values which independently define type and structure information for the data they carry. Furthermore, these elements and attributes model data and structure to a given software domain, for example c# programming language [20]. There is a rich library of XML tools, for example parsers, which act parsing, mapping and transforming generic XML data types for and from the given software domain. This is actually an essential aspect of Web services enabling the highly

desirable interoperability across software domains. Thus, programmers can transform XML data into application-readable format and vice-versa. The big difference between HTML and XML is that the former is designed to format and display data where the latter is a carrier of information designed to describe, store and transmit data with focus on what data is. It is also a self-descriptive language with not predefined tags.

As it has been mentioned before, XML Schema is a basic element of an XML document. It can be defined as a definition language that enables you to constrain conforming XML documents to a specific vocabulary and a specific hierarchical structure [11]. XML Schema is analogous to a database schema, which defines the column names and data types in database tables.

XML meta data is a form of description. It describes the purpose or meaning of raw data values via a text format to more easily enable exchange, interoperability, and application independence. As description, the general rule applies that "more is better." Meta data increases the fidelity and granularity of our data [11].

There are several alternatives for XML messaging. For example, you could use XML Remote Procedure Calls (XML-RPC) or SOAP or just HTTP GET/POST and pass arbitrary XML documents. This options are illustrated in the following figure:

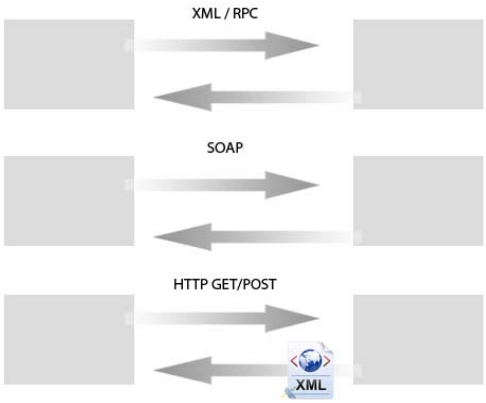


Figure 10: Forms of XML messages

Describe data types by using the XML Schema Definition language (XSD): XML Schemas are an application of XML used to express the structure of XML documents.

Web Service Description Language (WSDL)

Since SOAP specification does not address description, WSDL constitutes the standard specification for the service description layer within the web service protocol Stack.

WSDL is an XML grammar which provides the common communication tunnel (language) between a service provider and a service consumer. WSDL contains a set of definitions that fully describe the deployed web service by specifying a public interface for it. In this public

interface lies all information about the operations of the service, the data type of messages it accepts and generates, the communication protocols it uses and where it resides as well. WSDL can be considered a tool which automates the machine-to-machine communication and in the case of SOAP services it includes built-in extensions for defining them. Anybody given a WSDL file, has all the information needed to create a client application that can access the specific Web Service. Most of the times, WSDL is generated automatically from existing application components by software development kits (SDKs). Some of the strengths of WSDL include their capacity to provide a structured approach of defining web service interfaces and on the same time providing less code for client applications to implement.

This is how a WSDL document looks like:

```
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:soap="
http://schemas.xmlsoap.org/wsdl/soap/" xmlns:soapenc="http://schemas.xmlsoap
.org/soap/encoding/"xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-utility-
1.0.xsd" xmlns:xsd="http://www.w3.org/2001/XMLSchema"xmlns:soap12="http://sc
hemas.xmlsoap.org/wsdl/soap12/" xmlns:tns="http://tempuri.org/" xmlns:wsa="h
ttp://schemas.xmlsoap.org/ws/2004/08/addressing"xmlns:wsp="http://schemas.xm
lsoap.org/ws/2004/09/policy" xmlns:wsap="http://schemas.xmlsoap.org/ws/2004/
08/addressing/policy"xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
xmlns:msc="http://schemas.microsoft.com/ws/2005/12/wsdl/contract" xmlns:wsa1
0="http://www.w3.org/2005/08/addressing"xmlns:wsx="http://schemas.xmlsoap.or
g/ws/2004/09/mex" xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
 name="MainWcfService" targetNamespace="http://tempuri.org/">
<wsdl:types>
<xsd:schema targetNamespace="http://tempuri.org/Imports">
<xsd:import schemaLocation="http://85.72.56.17:81/MainWcfService.svc?xsd=xsd
0" namespace="http://tempuri.org/"/>
<xsd:import schemaLocation="http://85.72.56.17:81/MainWcfService.svc?xsd=xsd
1" namespace="http://schemas.microsoft.com/2003/10/Serialization/"/>
<xsd:import schemaLocation="http://85.72.56.17:81/MainWcfService.svc?xsd=xsd
2" namespace="http://schemas.datacontract.org/2004/07/MarkClassLibrary"/>
</xsd:schema>
</wsdl:types>
<wsdl:message name="IMainWcfService_GetModelsByPriceRange_InputMessage">
<wsdl:part name="parameters" element="tns:GetModelsByPriceRange"/>
</wsdl:message>
<wsdl:message name="IMainWcfService_GetModelsByPriceRange_OutputMessage">
<wsdl:part name="parameters" element="tns:GetModelsByPriceRangeResponse"/>
</wsdl:message>
<wsdl:message name="IMainWcfService_GetPriceAndCompanyByModel_InputMessage">
<wsdl:part name="parameters" element="tns:GetPriceAndCompanyByModel"/>
</wsdl:message>
<wsdl:message name="IMainWcfService_GetPriceAndCompanyByModel_OutputMessage"
>
<wsdl:part name="parameters" element="tns:GetPriceAndCompanyByModelResponse"
/>
</wsdl:message>
<wsdl:portType name="IMainWcfService">
<wsdl:operation name="GetModelsByPriceRange">
<wsdl:input wsaw:Action="http://tempuri.org/IMainWcfService/GetModelsByPrice
Range" message="tns:IMainWcfService_GetModelsByPriceRange_InputMessage"/>
<wsdl:output wsaw:Action="http://tempuri.org/IMainWcfService/GetModelsByPric
eRangeResponse" message="tns:IMainWcfService_GetModelsByPriceRange_OutputMes
sage"/>
</wsdl:operation>
<wsdl:operation name="GetPriceAndCompanyByModel">
<wsdl:input wsaw:Action="http://tempuri.org/IMainWcfService/GetPriceAndCompa
nyByModel" message="tns:IMainWcfService_GetPriceAndCompanyByModel_InputMessa
ge"/>
```

```
<wsdl:output wsaw:Action="http://tempuri.org/IMainWcfService/GetPriceAndComp
anyByModelResponse" message="tns:IMainWcfService_GetPriceAndCompanyByModel_O
utputMessage"/>
</wsdl:operation>
</wsdl:portType>
<wsdl:binding name="BasicHttpBinding_IMainWcfService" type="tns:IMainWcfServ
ice">
<soap:binding transport="http://schemas.xmlsoap.org/soap/http"/>
<wsdl:operation name="GetModelsByPriceRange">
<soap:operation soapAction="http://tempuri.org/IMainWcfService/GetModelsByPr
iceRange" style="document"/>
<wsdl:input>
<soap:body use="literal"/>
</wsdl:input>
<wsdl:output>
<soap:body use="literal"/>
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="GetPriceAndCompanyByModel">
<soap:operation soapAction="http://tempuri.org/IMainWcfService/GetPriceAndCo
mpanyByModel" style="document"/>
<wsdl:input>
<soap:body use="literal"/>
</wsdl:input>
<wsdl:output>
<soap:body use="literal"/>
</wsdl:output>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="MainWcfService">
<wsdl:port name="BasicHttpBinding_IMainWcfService" binding="tns:BasicHttpBin
ding_IMainWcfService">
<soap:address location="http://85.72.56.17:81/MainWcfService.svc"/>
</wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

Now the WSDL document is going to be analyzed according to the basic elements from which it is structured:

<**Definitions**> : The element “Definitions” is the "root" element of a WSDL document. It specifies the name of the Web service, declares multiple namespaces used throughout the document, and contains all the other elements of the service.

```
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:soap="
http://schemas.xmlsoap.org/wsdl/soap/" xmlns:soapenc="http://schemas.xmlsoap
.org/soap/encoding/"xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-utility-
1.0.xsd" xmlns:xsd="http://www.w3.org/2001/XMLSchema"xmlns:soap12="http://sc
hemas.xmlsoap.org/wsdl/soap12/" xmlns:tns="http://tempuri.org/" xmlns:wsa="h
ttp://schemas.xmlsoap.org/ws/2004/08/addressing"xmlns:wsp="http://schemas.xm
lsoap.org/ws/2004/09/policy" xmlns:wsap="http://schemas.xmlsoap.org/ws/2004/
08/addressing/policy"xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
xmlns:msc="http://schemas.microsoft.com/ws/2005/12/wsdl/contract" xmlns:wsa1
0="http://www.w3.org/2005/08/addressing"xmlns:wsx="http://schemas.xmlsoap.or
g/ws/2004/09/mex" xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
 name="MainWcfService" targetNamespace="http://tempuri.org/">
<wsdl:types>...</wsdl:types>
<wsdl:message name="IMainWcfService_GetModelsByPriceRange_InputMessage">...<
/wsdl:message>
<wsdl:message name="IMainWcfService_GetModelsByPriceRange_OutputMessage">...
</wsdl:message>
<wsdl:message name="IMainWcfService_GetPriceAndCompanyByModel_InputMessage">
...</wsdl:message>
```

```
<wsdl:message name="IMainWcfService_GetPriceAndCompanyByModel_OutputMessage"
>...</wsdl:message>
<wsdl:portType name="IMainWcfService">...</wsdl:portType>
<wsdl:binding name="BasicHttpBinding_IMainWcfService" type="tns:IMainWcfServ
ice">...</wsdl:binding>
<wsdl:service name="MainWcfService">...</wsdl:service>
</wsdl:definitions>
```

<**Types**> : The element "Types" describes all types of data[1] used for the server - client exchange. WSDL uses the W3C XML Schema, by default. This element is not required in case the service uses only simple types of XML Schema, such as string and integer.

```
<wsdl:types>
<xsd:schema targetNamespace="http://tempuri.org/Imports">
<xsd:import schemaLocati
on="http://85.72.56.17:81/MainWcfService.svc?xsd=xsd0" namespace="http://tem
puri.org/"/>
<xsd:import schemaLocation="http://85.72.56.17:81/MainWcfService.svc?xsd=xsd
1" namespace="http://schemas.microsoft.com/2003/10/Serialization/"/>
<xsd:import schemaLocation="http://85.72.56.17:81/MainWcfService.svc?xsd=xsd
2" namespace="http://schemas.datacontract.org/2004/07/MarkClassLibrary"/>
</xsd:schema>
</wsdl:types>
```

<**Message**> : The "Message" element is the basic communication element of SOAP. It describes a one-way message, whether for a SOAP request or a response[2]. It specifies the name of the message and it can contain more than one sub-elements called «part», which can refer either to parameters or return values.

```
<wsdl:message name="IMainWcfService_GetModelsByPriceRange_InputMessage">
<wsdl:part name="parameters" element="tns:GetModelsByPriceRange"/>
</wsdl:message>
<wsdl:message name="IMainWcfService_GetModelsByPriceRange_OutputMessage">
<wsdl:part name="parameters" element="tns:GetModelsByPriceRangeResponse"/>
</wsdl:message>
```

<**PortType**> : "PortType" is the most important WSDL element which describes the set of operations supported by the service, the messages that are involved and the connection point to the service. If each operation can be compared to a function in a traditional programming language then "PortType" is the function library (or class) in this language. A Web Service can have multiple interfaces represented by different portTypes.

```
<wsdl:portType name="IMainWcfService">
<wsdl:operation name="GetModelsByPriceRange">
<wsdl:input wsaw:Action="http://tempuri.org/IMainWcfService/GetModelsByPrice
Range" message="tns:IMainWcfService_GetModelsByPriceRange_InputMessage"/>
<wsdl:output wsaw:Action="http://tempuri.org/IMainWcfService/GetModelsByPric
eRangeResponse" message="tns:IMainWcfService_GetModelsByPriceRange_OutputMes
sage"/>
</wsdl:operation>
```

[1] The data types used by the operations of the web service

[2] The request-response type is the most common operation type, but WSDL defines three more types: one - way, solicit - response, notification.

```
<wsdl:operation name="GetPriceAndCompanyByModel">
<wsdl:input wsaw:Action="http://tempuri.org/IMainWcfService/GetPriceAndCompa
nyByModel" message="tns:IMainWcfService_GetPriceAndCompanyByModel_InputMessa
ge"/>
<wsdl:output wsaw:Action="http://tempuri.org/IMainWcfService/GetPriceAndComp
anyByModelResponse" message="tns:IMainWcfService_GetPriceAndCompanyByModel_O
utputMessage"/>
</wsdl:operation>
</wsdl:portType>
```

<**Binding**> : The “Binding” element describes the specific details (e.g message format, communication protocols) of how the service will be implemented in the network. It actually binds a porType to a specific communication protocol, in our case SOAP over HTTP.

```
<wsdl:binding name="BasicHttpBinding_IMainWcfService" type="tns:IMainWcfServ
ice">
<soap:binding transport="http://schemas.xmlsoap.org/soap/http"/>
<wsdl:operation name="GetModelsByPriceRange">...</wsdl:operation>
<wsdl:operation name="GetPriceAndCompanyByModel">...</wsdl:operation>
</wsdl:binding>
```

<**Service**> : The” Service” element pulls together the portType, the binding and the location (URI) of the web service. It is usually a URL which points to the server where the service is implemented.

```
Example:
<wsdl:service name="MainWcfService">
<wsdl:port name="BasicHttpBinding_IMainWcfService" binding="tns:BasicHttpBin
ding_IMainWcfService">
<soap:address location="http://85.72.56.17:81/MainWcfService.svc"/>
</wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

Universal Description, Discovery and Integration (UDDI)

Universal Description, Discovery, and Integration (UDDI) is one of the most widely recognized discovery mechanisms. Its role is to assist in discovering what services exist in the world.

UDDI is a consortium of companies (industry and business leaders such as Microsoft and IBM) formatting a partnership in order to promote interoperability and adoption of web services. UDDI is consisted of several internet-based business registries working similar to DNS. Each registry works similar to yellow pages allowing businesses to describe themselves as business entities publicly listing their description but also the services they offer in XML format. UDDI can be considered as a business centric view of the web. A difference from LDAP is that LDAP describes anything unlike UDDI which only describes business entities.
The same set of protocols work internally as well (e.g a company’s intranet) enabling the creation of local or private UDDIs where many departments can use in order to leverage their work.

UDDI architecture:

UDDI Data Model or structures: This is an XML Schema for describing businesses and web services including four types of information (elements): businessEntity which includes information about the actual business, businessService which includes information about the services, bindingTemplate which includes information about the location of the service and how to access it and finally tModel providing pointers to external technical specifications.

UDDI API: In order to access a UDDI registry[1], there are provided two SOAP interfaces. The first one is called PublishSOAP used by service providers to list their services. The second one is called InquireSOAP used by service consumers to find and retrieve a service

UDDI Cloud Services: Operator sites that provide implementations of the UDDI specification and synchronize all data on a scheduled basis.

The basic steps for service providers to publish their services are the following:

1. Each service provider is registered as a unique UDDI business entity specifying the categories and identifiers applied to this entity
2. The service provider registers each of the services it wants to offer as a UDDI business service specifying again the categories applied to this specific service entity
3. The final step is registering the implementation details of the web service(s) paying attention to the location details of the deploying service.

UDDI contents:

Information

White pages: Contain typical information such as name, address, telephone and other business contact information.

Yellow pages: Provide information about categorizing businesses based on existing (not electronic) categorization standards.

Green pages: In green pages businesses actually list the services they have to offer to the world. Green pages contain all the technical details for machine-to-machine communication in order programs to consume the services.

Functions

Publish: How a web service provider has registered its entity and services.

[1] Access either for searching or publishing

Find: How an application finds a particular web service.

Bind: How an application connects and interacts with a web service

Hypertext Transfer Protocol (HTTP)

Hypertext Transport Protocol (HTTP) is the most popular option for service transport. It is very popular protocol that exists long before the advent of Web Services[1]. Its goal is to facilitate the transfer of requests from a browser to a Web server allowing communication[2] through corporate firewalls. Web services takes advantage of this mature protocol to move SOAP messages and WSDL documents from one computer to another although other protocols such as FTP and SMTP can be used to perform this same function. However, HTTP remains the most extensively used transport protocol for Web Services.

2.4 Governing bodies

Various governing bodies have been created from the beginning of Web Service adoption in order to manage the decision-making process of refining Web Service technology and its specifications. However, most of these efforts where lucking objectivity, because where running under the authority of specific vendors where trying to obtain most of the benefits of this new technology and setting their own terms to their competitors. Over the years, some non-profit consortiums were created in order to control industry competition but also to set a common basis for the development and adoption of Web Service technology. There are three non-profit organizations set as the main governing bodies:

2.4.1 The World Wide Web Consortium

It is considered probably the most important organization managing the core Web Service technology specifications such as SOAP, XML, WSDL, HTTP.

W3C develops interoperable technologies (specifications, guidelines, software, and tools) to lead the Web to its full potential. W3C is a forum for information, commerce, communication, and collective understanding
Source : [http://www.w3.org/2006/07/privacy-ws/papers/29-wenning-position/]

[1] Nowadays, HTTP is so well-known and widely used that identical to the Web. Therefore, it is not going to be given much attention on this protocol.

[2] SOAP messages are masquerade as HTTP messages

2.4.2 Organization for the Advancement of Structured Information Standards

OASIS, founded in 1993, is a not-for-profit, global consortium that drives the development, convergence, and adoption of open standards for the global information society.
Source: [http://www.oasis-open.org/org]

OASIS has the precedence of providing Web Services standards especially for security (WS-Security, SAML specifications). Except for security standards, OASIS took over UDDI after is ascertained its acceptance.
The reader will find in OASIS two of the most widely respected information portals on XML and Web services standards: Cover Pages and XML.org.

2.4.3 Web Services Interoperability organization

WS-I, now part of OASIS, started as an open industry effort, a consortium of vendors and end-user organizations, participating in some of the industry leaders such as Microsoft, IBM, Oracle, SAP and HP. Their target was to establish a series of blue-prints, or in other words industry best practices, for Web Services' adoption in order to achieve Interoperability across platforms, applications and programming languages. By setting a standards integrator, WS-I dreamed Web Services Interoperability which in turn would bring acceleration of Web Services deployment encouraging their adoption. The benefits where numerous and therefore WS-I started working on setting profiles (specifications, guidelines and conventions), sample applications (use cases/usage scenarios, sample code and applications) as well as test tools and supporting materials (profile conformance tools, documentation and white papers). Unfortunately, W3C and OASIS where working simultaneously addressing the same problem by taking a different approach.

By the end of 2010, WS-I absorbed by OASIS. The press did not welcome this news criticizing that the end of WS-* family specifications is inevitable destined purely for maintenance at OASIS.

PART II WCF Web Service: Design and development

Introduction

My initial intention was to develop a simple SOAP Web Service connected with a database. Since I am not an expert in software development is short, I started learning how to use modern IDEs (Integrate Development Environment) such as NetBeans 7.1 and Visual Studio 2010 Express. Both are excellent programs offering great set of tools enabling even beginners to software development to write and modify their own code. IDEs provide also tools for detecting code errors and correcting them. Some of the many features that IDEs include are the following: compiler, editor, linker, debugger, assembler, libraries, and a variety of other tools designed to work together etc.

IDEs support a lot of programming languages. However, NetBeans is more Java oriented (also supporting C/C++). On the other hand Visual Studio supports Visual Basic, C# etc. and since I have little knowledge on C#, I decided to use Visual Studio as my Development Kit.

3.1 Implementation environment

Development tools

DBMS: MS SQL Server 2008 R2 express edition version 10.50.1617.0

Microsoft Visual Studio 2010 Express

Programming Language: C#

JavaScript library: jQuery 1.7.1

Data – interchange format: JSON

Web Service was hosted in an IBM Server with the following characteristics:

OS: Microsoft Windows Server 2003 Standard Edition SP2
CPU: Intel E5310 1.6 GHz
RAM: 4GB

DBMS: MS SQL Server 2008 R2 express edition version 10.50.1617.0

Microsoft .NET Framework 3.5 SP1 (for the installation of SQL DBMS)
Microsoft .NET Framework 4 Extended (in which the service was created)
Microsoft XML Core Services 6.0 Service Pack 2 (MSXML 6 SP2)
Microsoft ASP.NET MVC 3

JAVA SDK, SE v.1.4.2_05
Java 2 Runtime environment, SE v.1.4.2_05

Internet Information Services 6.0

HTTP browser: Mozilla Firefox 12.0

Hardware Firewall: Zyxel P-662H-D3

3.1.1 Windows Communication Framework (WCF)

Windows Communication Framework (WCF) is the answer of Microsoft to the cloud and service-oriented communication. It is a unified[1], cross-domain enabled, architecture which provides a set of libraries and tools for easy creation of web services and clients to access those services. Wherever the service and client run[2] they can interact via various ways such as SOAP or binary protocol. The services follow the WCF Web Services Description Language interface, so it can be used by any WCF client regardless of the system they are hosted. WCF implements a "many to many" model, meaning that one service can be used by multiple and different types of clients (for example SOAP and binary at the same time) and one client can use multiple services.

Some of the advantages WCF offers are Interoperability, reliability and security compared to the classic .Net (ASMX) Web Services. WCF is also powerful and flexible enabling small changes in the configuration instead of writing code in case for changing, for example, the security and logging mechanisms or communication binding. In addition, providing an extra security feature, by default, WCF do not exposes metadata.

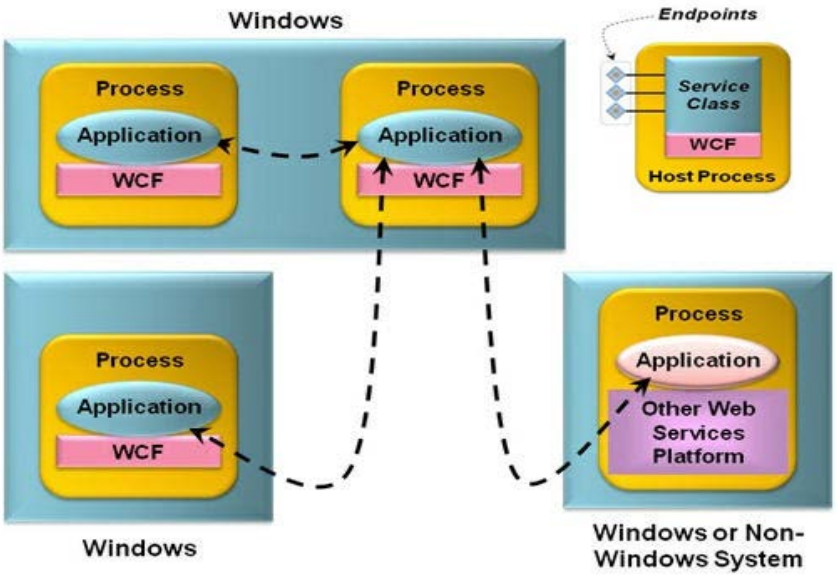


Figure 11: How WCF works (source: msdn)

[1] WCF unified the original .NET Framework communication technologies. WCF is included in the .NET Framework 4

[2] WCF doesn't define a required host

3.1.2 Microsoft .NET Framework

Microsoft .NET Framework is an integral Windows component and software framework which supports building and running applications and XML Web services. It consists of two major components: the Common Language Runtime (CLR) and the Class Library component. Microsoft .NET Framework also includes the ASP.NET [1]SOAP extension framework which allows ASP.NET components to process SOAP messages. The following figure illustrates the .NET Framework in context.

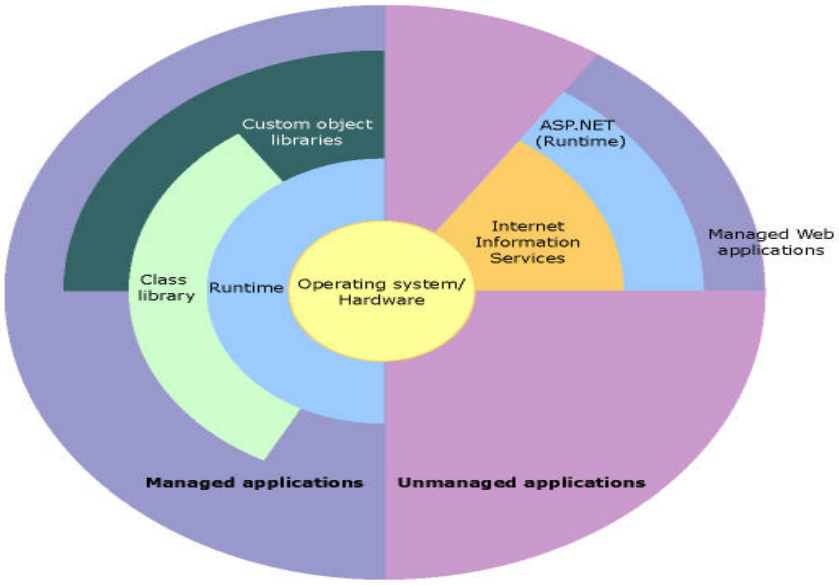


Figure 12: Microsoft .NET Framework in context (source: msdn[2])

3.1.3 Internet Information Services (IIS)

Internet Information Services (IIS) or Internet Information Server is a web server3 Microsoft infrastructure which plays the role of XML Web Service provider by supporting and hosting XML Web Services. IIS provides an HTTP listener for Web Services developed in .NET Framework and Visual Studio.

[1] ASP.NET is a Web application framework developed and marketed by Microsoft to allow programmers to build dynamic Web sites, Web applications and Web services

[2] http://msdn.microsoft.com/library/zw4w595w(VS.100).aspx

[3] IIS is an integral part of Windows Server products

IIS can invoke a service on behalf of a client, by using many different options. A Web server can start a Common Gateway Interface (CGI) application; run a script interpreter as done in Microsoft Active Server Pages (ASP), or invoke an ISAPI application.
When IIS works in conjunction with the common language runtime, it uses an ISAPI filter to intercept requests for pages with the extension .asmx, and then start a run-time host. The run-time host then executes the code for an XML Web service that is implemented by using the .NET Framework [6].

The .NET extensions to IIS recognize files ending in .asmx as web services and automatically export the functions of the referenced assemblies.

3.2 Designing the Web Service

The "business" logic of the service is to query a database about laptop characteristics (e.g models, price) using AJAX calls. The Service (its WSDL) is not using any UDDI registry (private or not) for being advertised. Instead it is published through the web server which hosts the service. One assumption made is that the service is offered to limited users (during its beta phase) who have authenticated access on it through Integrated Windows Authentication.

Users' target group is home PC users and thus a client application was created to provide users with a friendly User Interface (UI) in order to consume the service. The UI (from now on "application") is hosted by the same web server which hosts the initial Web Service.

Therefore, we have the following information at the moment:

Web Service itself published through IIS in TCP port 81

Full Uri path of WSDL: http://85.**.**.**:81:/MainWcfServices.svc?wsdl

Web Service Application (UI) installed on IIs in TCP port 82.

Full Url path: Http://85.**.**.**:82

Next, I will present the steps I followed to create the Web Service.

First step: Data retrieving

To begin with, I used MS SQL to create a database called laptops. This database contains two basic tables: laptop_models and laptop_characteristics. Next figure illustrates the fields of each table and how these tables are connected with each other.

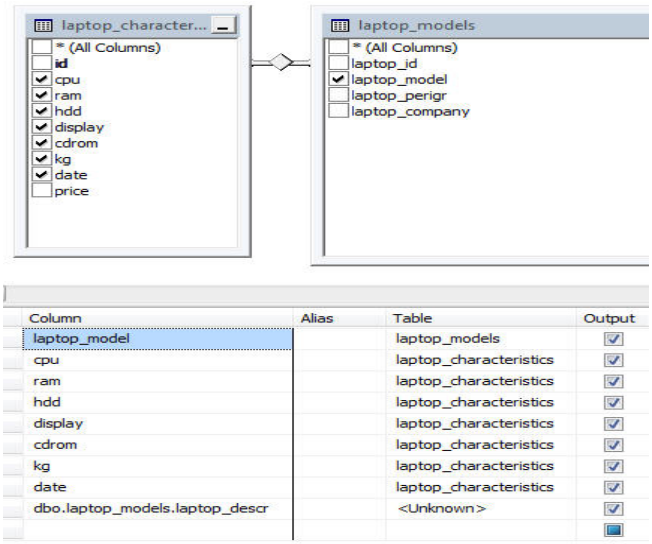


Figure 13: A view of "laptops" database tables

Instead of putting queries inside the service code, I decided to create and use two Stored Procedures. This is making my life easier since every time I want to change the queries (for example adding or removing a field) I just connect to the database and change the specific Stored Procedure, not the service's code. As a result, there are two basic Stored Procedures retrieving data from the database tables:

A) StoredProcedure [dbo].[GetListOfModelsByPrice]

```
USE [laptops]
GO

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO


ALTER PROCEDURE [dbo].[GetListOfModelsByPrice]
	@priceRangeID int
AS
BEGIN

	IF @priceRangeID=1
		BEGIN
			select LM.laptop_id,
				LM.laptop_model
			from	Laptops.dbo.laptop_models as LM,
				Laptops.dbo.laptop_characteristics AS LC
			WHERE LM.laptop_id=LC.id AND LC.price <=500
		END
	ELSE IF @priceRangeID=2
		BEGIN
			select LM.laptop_id,
				LM.laptop_model
			from	Laptops.dbo.laptop_models as LM,
				Laptops.dbo.laptop_characteristics AS LC
			WHERE LM.laptop_id=LC.id AND LC.price > 500 AND LC.price <=1000
		END
	ELSE IF @priceRangeID=3
		BEGIN
			select LM.laptop_id,
				LM.laptop_model
			from	Laptops.dbo.laptop_models as LM,
```

```
                                Laptops.dbo.laptop_characteristics AS LC
                        WHERE  LM.laptop_id=LC.id AND LC.price >1000
                END


END
```

This is the first Stored Procedure where I define some price ranges with fixed values. According to the price range I choose, I get a list of models. Unless I choose one, I cannot continue to use the second Stored Procedure which follows.

B) StoredProcedure [dbo].[GetPriceCompanyValuesByModelID]

```
USE [laptops]
GO

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

ALTER PROCEDURE [dbo].[GetPriceCompanyValuesByModelID]
        @modelID INT
AS
BEGIN
        select LM.laptop_company,
                                LC.price
        from    Laptops.dbo.laptop_models as LM,
                        Laptops.dbo.laptop_characteristics AS LC
        WHERE LM.laptop_id = @modelID AND LM.laptop_id=LC.id
END
```

The above Stored Procedure retrieves price and company providing the model.
Finishing the first step, I have completed retrieving raw data from the database.

Second step: Representing data to the service

In this step I started writing the code. By retrieving raw data from the first step, I must represent with data in such structures that the service itself can read. This is happening through C# classes/objects. In the IMainWcfService.cs file presented below, I define the methods (public interface) for the input and output. We notice the I letter before the name which implies the Interface.
The first method, operation contract: “GetModelsByPriceRange” waits for an integer in order to give back a list of model Objects. List<ModelObj> is actually an array, a structure which retains many similar data (e.g laptop models). Let’s see the code of the class ModelObj:

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Runtime.Serialization;
using System.Text;
using DBManager;

namespace MarkClassLibrary
{
    [Serializable]
    [DataContract(Name = "ModelObj")]
    public class ModelObj
```

```
    {
        [DataMember(Name = "ModelID")]
        public int ModelID { get; set; }

        [DataMember(Name = "ModelDescription")]
        public string ModelDescription { get; set; }

        public static List<ModelObj> GetListOfModelsByPrice(int priceRange, string
connString)
        {
            var listOfModelsByPrice = new List<ModelObj>();
            var db = new DbManager(DataProvider.SqlServer, connString);

            try
            {
                db.Open();
                db.CreateParameters(1);
                db.AddParameters(0, "priceRangeID", priceRange);
                const string sql = @"[dbo].[GetListOfModelsByPrice]";
                DataSet ds = db.ExecuteDataSet(sql, CommandType.StoredProcedure);

                if (ds != null && ds.Tables.Count > 0 && ds.Tables[0] != null)
                {
                    var dt = ds.Tables[0];
                    if (dt.Rows != null && dt.Rows.Count > 0)
                    {
                        listOfModelsByPrice = (from DataRow r in dt.Rows
                                               select new ModelObj()
                                               {
                                                   ModelDescription =
r["laptop_model"].ToString(),
                                                   ModelID =
Convert.ToInt32(r["laptop_id"].ToString())

                                               }).ToList();


                    }
                }

            }
            catch (Exception ex)
            {
                throw new Exception(ex.ToString());
            }
            finally
            {
                if (db.Connection != null)
                {
                    db.Close();
                    db.Dispose();
                }
            }


            return listOfModelsByPrice;
        }

    }
}
```

As we can see, this class represents models. It waits for two types of values: integer for ModelID and string for ModelDescription. It needs as parameter a priceRangeID.

Moving to the next method, operation contract: “GetPriceAndCompanyByModel” the array here is PriceCompanyObj classs. Let’s examine its code:

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Runtime.Serialization;
```

```
using System.Text;
using DBManager;

namespace MarkClassLibrary
{
    [Serializable]
    [DataContract(Name = "PriceCompanyObj")]
    public class PriceCompanyObj
    {
        [DataMember(Name = "Company")]
        public string Company { get; set; }

        [DataMember(Name = "Price")]
        public int Price { get; set; }


        public static PriceCompanyObj GetPriceCompanyValuesByModelID(int modelID,
string connString)
        {
            var priceCompanyValues = new PriceCompanyObj();

            var db = new DbManager(DataProvider.SqlServer, connString);

            try
            {
                db.Open();
                db.CreateParameters(1);
                db.AddParameters(0, "modelID", modelID);
                const string sql = @"[dbo].[GetPriceCompanyValuesByModelID]";
                DataSet ds = db.ExecuteDataSet(sql, CommandType.StoredProcedure);

                if (ds != null && ds.Tables.Count > 0 && ds.Tables[0] != null)
                {
                    var dt = ds.Tables[0];
                    if (dt.Rows != null && dt.Rows.Count > 0)
                    {
                        priceCompanyValues = (from DataRow r in dt.Rows
                                              select new PriceCompanyObj()
                                                  {
                                                      Company =
r["laptop_company"].ToString(),
                                                      Price =
Convert.ToInt32(r["price"].ToString())

                                                  }).FirstOrDefault();


                    }
                }

            }
            catch (Exception ex)
            {
                throw new Exception(ex.ToString());
            }
            finally
            {
                if (db.Connection != null)
                {
                    db.Close();
                    db.Dispose();
                }
            }

            return priceCompanyValues;
        }
    }
}
```

Similarly to the first class, PriceCompanyObj class accepts two type of values: string for Company and integer for Price. The parameter it needs is modelID.

In both classes we can notice their connection with the database, each one with its own Stored Procedures. To continue, we move on to the part where I implement all the methods (operations) I have described in IMainWcfService.cs . This part is MainWcfService.svc.cs , the actual Web Service. As will see inside the code, here we have the actual communication with the database (private readonly string). We get list of Models by invoking class ModelObj.GetListOfModelsByPrice (priceRange, _connString) and Price, Company values by invoking class PriceCompanyObj.GetPriceCompanyValuesByModelID(modelID, _connString)

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Activation;
using System.ServiceModel.Web;
using System.Text;
using MarkClassLibrary;

namespace MainWcfService
{
   [AspNetCompatibilityRequirements(RequirementsMode =
AspNetCompatibilityRequirementsMode.Allowed)]
    public class MainWcfService : IMainWcfService
    {
        private readonly string _connString =
ConfigurationManager.ConnectionStrings["connString"].ConnectionString;


        public List<ModelObj> GetModelsByPriceRange(int priceRange)
        {
            var listOfModels = new List<ModelObj>();
            try
            {
                listOfModels = ModelObj.GetListOfModelsByPrice(priceRange,
_connString);
            }
            catch (Exception ex)
            {

                throw new Exception(ex.ToString());
            }

            return listOfModels;
        }

        public PriceCompanyObj GetPriceAndCompanyByModel(int modelID)
        {
            var priceCompanyValues = new PriceCompanyObj();
            try
            {
                priceCompanyValues =
PriceCompanyObj.GetPriceCompanyValuesByModelID(modelID, _connString);
            }
            catch (Exception ex)
            {

                throw new Exception(ex.ToString());
            }

            return priceCompanyValues;
        }
    }
}
```

Note: The connection String parameters for the connection with the database, reside inside Web.config file, enabling us to change/ move the database even in another host, domain or network:

```
<connectionStrings>
    <add name="connString" connectionString="Data
Source=PC\SQLSERVER2008R2;MultipleActiveResultSets=True;uid=sa;pwd=***;database=Laptop
s;Connection timeout=180;" providerName="System.Data.SqlClient" />

</connectionStrings>
```

Step 3: Generating the client

Avoiding the details of creating a web application, I created a client application (a User Interface) in C# to consume the service I have just described. This client is named MainWcfServiceClient. In this point i would like to notice the need of adding the web service as a service reference to the client application in order for the application to know the location of the service it is going to consume. This resides inside the application code under the application's Web.config file:

```
<client>
      <endpoint address="http://localhost/MainWcfService/MainWcfService.svc"
binding="basicHttpBinding" bindingConfiguration="BasicHttpBinding_IMainWcfService"
contract="MainWcfService.IMainWcfService" name="BasicHttpBinding_IMainWcfService" />
</client>
```

Step 4: Routing queries and data

The client application (UI) uses a proxy object to facilitate the communication with the Web Service. This proxy is called WebHandler. It is an intermediary object between the service and the UI. The most important element of the WebHandle is called ProcessRequest which is actually the heart of this proxy.
At request time, the proxy accepts, through jQuery, all AJAX calls (HTTpRequests) from the application. It then converts each call in C# objects in order for the service to understand the requests and retrieve data from the database.
At response time, we have the opposite direction. The proxy transforms raw data that the service has retrieved from the database into C# format and through a JavaScript Serializer translates this information into java. Then returns the results back to the client UI using JSON (an XML schema). Finally, the UI represents the information to the end user into HTTL format.

Looking at the code in WebHandler.ashx.cs file will help reader capture the idea:

```
using System;
using System.Web;
using System.Web.Script.Serialization;
using MarkApp.MainWcfService;

namespace MarkApp
{

    public class WebHandler : IHttpHandler
    {
        private const string ActionID = "actionID";
        private const string PriceID = "priceID";
        private const string ModelID = "modelID";
```

```
        public bool IsReusable
        {
            get
            {
                return false;
            }
        }

        public void ProcessRequest(HttpContext context)
        {
            HttpRequest request = context.Request;
            var actionID = Convert.ToInt32(request[ActionID]);
            var priceID = Convert.ToInt32(request[PriceID]);
            var modelID = Convert.ToInt32(request[ModelID]);

            MainWcfServiceClient serviceClient = new MainWcfServiceClient();
            var sr = new JavaScriptSerializer();

            if(actionID == 1)
            {

                var listOfModels = serviceClient.GetModelsByPriceRange(priceID);
                HttpContext.Current.Response.Write(sr.Serialize(listOfModels));
            }
            else if(actionID == 2)
            {
                var priceAndTypeValues =
serviceClient.GetPriceAndCompanyByModel(modelID);
                HttpContext.Current.Response.Write(sr.Serialize(priceAndTypeValues));
            }



        }


    }
}
```

“ProcessRequest” is the most important element of WebHandler. All calls are routed to / from here. The proxy receives data from the client (UI) and responds back always in JSON format.

PART III Web Services Security

Introduction

In our challenging and demanding Information Society era, security is always a major concern for every one of us regardless the role we have (consumers, providers or intermediaries of a service). Each time we examine security, we must not forget the general requirements that every Security Framework fulfills:

a) Authentication
b) Authorization
c) Confidentiality
d) Integrity

Authentication deals with identity. Who is who and how can she prove it?

Authorization deals with the access control. What someone can access or is authorized to do?

Confidentiality is mostly about privacy. Encryption warrantees that data will be viewed only by authorized persons.

Integrity concerns tamper – proofing. It insures that the message should not be tampered with at any way until it reaches its destination.

Furthermore, there is another security principle defined as a major business requirement. This is the non-repudiation principle notably used in business context for proving what has really happened.

4.1 Security Considerations

"Web Services security is a result of many activities"

Security is a complex topic but critical to Web Services adoption. Considering the fact that none of core Web Service technologies, such as SOAP or XML, include any security mechanisms as well as the numerous threats that exist or arise by the mass adoption of Web Services, it becomes crucial for setting and utilizing security parameters for every Web Service implementation. Therefore, security is fundamental to the adoption and development of Web Services.

Fortunately, nowadays, there are a lot of security specifications adding security capabilities to the technologies that make web services happen. WS-I Specifications, now governed by OASIS, provide a set of Web Services security specifications. These specifications are evolving and change over time. The most important are the following:

WS-Security, WS-Policy, WS-Trust, WS-Privacy, WS-SecureConversation, WS-Federation and WS-Authorization.

Specification	Description
WS-Security	Describes how to attach security tokens (certificates and Kerberos tickets), signatures, and encryption headers to SOAP messages.
WS-Policy	Describes the features and limitations of the security policies on intermediaries and endpoints.
WS-Trust	Describes a framework for secure interoperation of trust models between Web services.
WS-Privacy	Describes how both Web services and their requestors specify subject privacy preferences and organizational privacy practice statements.
WS-SecureConversation	Describes how to authenticate and manage message exchange between parties.
WS-Federation	Describes how to manage and broker the trust relationships in a heterogeneous federated environment.
WS-Authorization	Describes how to manage authorization data and policies.

Figure 14: OASIS WS-I Specifications

WS-Security, now replaced by OASIS WSS TC, is a fundamental layer of Web Services Security examining security between SOAP message exchanges. Its goal is to provide end-to-end security. It supports a variety of security models by describing how to provide integrity, confidentiality, and message authentication through the attachment of security tokens (custom tokens, certificates and Kerberos tickets), SAML assertions, signatures, and encryption headers to SOAP messages. This is further addressed by the following specifications according to WS-Security 1.1[1] OASIS standard:

WS-Security Core Specification 1.1 or WS-Security SOAP Message Security 1.1
Username Token Profile 1.1
X.509 Token Profile 1.1
SAML Token profile 1.1
Kerberos Token Profile 1.1
Rights Expression Language (REL) Token Profile 1.1
SOAP with Attachments (SWA) Profile 1.1

Figure 15: Web services security architecture

[1] WS-Security 1.1 OASIS Standard replaced version 1.0

WS-Trust is another fundamental layer describing a framework for establishing trust relationships (direct or through third-parties) for secure interoperation between Web Services. It uses security-token issuance services which are transferred through WS-Security to ensure their confidentiality and integrity.

WS-Privacy states and indicates conformance of incoming requests to organizational privacy policies and is usually combined with WS-Security and WS-Trust.

WS-SecureConversation describes how to authenticate and manage message exchange between parties. It uses security tokens that use symmetric (shared-secret) or asymmetric (public/private) key algorithms, session keys and per-message keys. WS-SecureConversation can be used in conjunction with WS-Security and transport layer security mechanisms.

WS-Federation describes the way of defining and managing trust relationships between service providers. Trust relationships will, of course, use all the previous specifications.

WS-Authorization describes how Web Service authorization is managed across different security providers with different authorization format and language.

A newcomer in Web Services Security will be confused at the beginning, encountering all these security specifications. Therefore, my intention is to provide a more clear view of Web Services Security by clarifying security mechanisms through layers. My analysis is more close to a Service Provider's scope of what can be done to secure the service, how clients will be authenticated and authorized, how privacy and data integrity will be preserved. Web Services Security is the result of many activities. As a result, my proposal is to handle security in layers, similar to the Web Service Protocol Stack. Special focus will be placed in the second layer concerning message security which directly affects Web Services. Moving to that direction we have the following basic layers:

1) Transport Layer
2) Message Layer
3) Application Layer

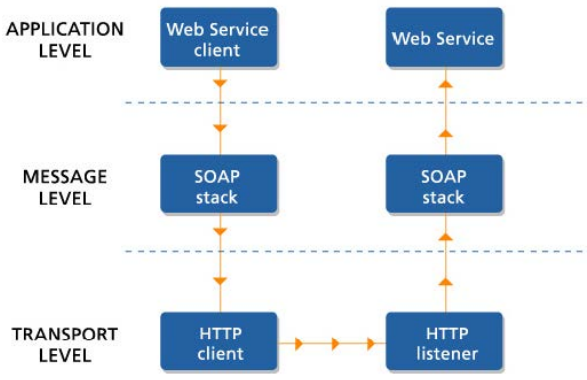


Figure 16: Web Services Security Stack

Every layer must be protected autonomously and / or in conjunction with the other layers. The methods used for each of these layers vary depending upon service provider's needs and the purpose an application serves.

During the next paragraphs, each of these layers will be analyzed but focus will be provided in data or message layer security.

4.1.1 Transport Layer security

Transport Layer uses existing Web tier technology such as HTTP and Secure Socket Layer[1] (SSL).

Authentication

Authentication mechanisms include HTTP authentication schemes – Basic or Digest and SSL client-side certificates.

Authorization

URL access control policies in the web tier provide security constraints for unauthorized users.

Confidentiality

Confidentiality includes SSL encrypted connections.

Integrity

Integrity often employs point-to-point SSL encryption to avoid data interception

As we can notice, Secure Sockets Layer is widely used in transport layer security. However, it accomplishes a piece of the security puzzle. Although SSL can be effective security mechanism between two endpoints such as a client-server environment, it is ineffective in multi-point scenarios. Such scenarios can include a number intermediary nodes routing or / and processing the data until it gets to the final endpoint. In such environments, even multiple SSL connections could not ensure security by user authentication and authorization. Furthermore, SSL cannot protect against several types of attacks such as buffer overflow or replay attacks. Moreover, SSL does require additional processing overhead if done in software causing system delays in case of extensive, multiple client access on the service. As a result, other technologies need to be used combined with or without SSL in order to accomplish security goals of Web services.

[1] SSL is a point-to-point data encryption protection used for mutual or one-way authentication

4.1.2 Message Layer Security

"Adding security to the message level is the best solution for Web services."

Message or Data Layer concerns data built in to the XML document usually as additional SOAP header fields. This layer is directly affecting the security of Web Services and definitely should not be overlooked even there has been implemented strong security in the other two layers. The specification from the WS-* family, that best addresses message security is SOAP message security v1.1. Inside this specification, the use of XML encryption and digital signatures, key management services and SAML, provide the security foundation for SOAP and any other XML-based messaging paradigms [13].

By developing authentication mechanisms on message layer, a Web Service can pass client credentials in the SOAP header or in the SOAP body as elements. Supposing we have a custom login authentication operation that returns a session key, the client would include the session identifier in the SOAP header. This description tries to present how trust integration comes true. This is the most difficult part in Web Services Security and can be overcome by outsourcing security to third-party trust services which guarantee the transactions between the service provider and service consumers. By invoking the appropriate trust service another Web Service could execute security functions such as encryption and signing.

Message Layer Security is a must, especially in case of multiple intermediaries between the client and the eventual Web Service.

Authentication

Examples of message authentication mechanisms are Single Sign-On (SSO) and SAML security tokens.

XML security credential standards such as SAML along with identity[1] management standards such as Passport and Liberty Alliance specifications. Both authentication and authorization are concerned about Identity management. Usually it includes security credentials such as username and password and X.509 certificates[2] as well.

Authorization

SSO, SAML and custom security tokens can provide both authentication and authorization mechanisms. In addition, Extensible Access Control Markup Language (XACML) defines standardized security access control using XML to state authorization rules over a public

[1] The concept of Identity is about who we are, how we can prove it and if so what are we allowed to do

[2] X.509 is a standard for public key infrastructure (PKI)

connection. XACML also allows validation and revocation, based on defined authorization rules

XML Key Management Specification (XKMS) is another security standard that details protocols for registration and distribution of public keys, so that the keys can be used in combination with XML digital signatures and encryption. XKMS was created to simplify the integration of digital certificates and public key infrastructure (PKI) with a multitude of applications. Applications that use this specification can easily integrate authentication, digital signature, and encryption services. XKMS includes support for certificate processing and revocation status checking.

XML Digital Signatures (XMLSIG) is one of the most widely used standards for XML Security. Although digital signatures were firstly used for non-repudiation purposes, today they offer more than that. One of the services they provide is authorization mechanisms by specifying the syntax and processing rules for an XML document to attach a digital signature in it (self-sign) . An XML digital signature takes data objects, calculates a digest (fixed-length representation of a variable-length stream), and places the result into the signature element. XML signatures can be attached to any form of digital content and can sign multiple types of resources, such as HTML and binary-encoded data (GIFs and JPEGs), a whole XML document or only specific sections of that document.

Confidentiality

Privacy could be achieved by encrypting[1] the messages. A custom encryption mechanism is the XML Encryption specification.

WS-Security leverages the XML encryption specification along with security tokens to keep portions of SOAP messages confidential. The encryption features can also support encryption technologies, processes, and operations by multiple parties. [13]

Integrity

As the distance between the original user and the eventual Web service gets bigger and bigger, integrity is more difficult to be maintained. In addition, many organizations do not want SOAP method invocations coming through their firewall if they are encrypted and cannot see them. Therefore, there is imperative need of protecting data being transferred across several intermediaries. XML Digital Signatures specification can assist in that provides more than authorization services. It ensures that data will not be altered and that has been originated from whoever has signed the message.

[1] *Encryption* is the act of taking data (usually referred to as clear text) and a short string (the key) and producing data (cipher text). The resulting cipher text is meaningless to a third party who does not know the key. Decryption is the inverse of encryption: taking cipher text and key and producing clear text.

4.1.3 Application Layer Security

This layer includes all security mechanisms provided for web application security. OWASP is the emerging standards body for web application security providing valuable information about major threats against web applications as well as countermeasures for these threats. Furthermore, OWASP provides excellent solutions for testing web applications.

Authentication

Application specific authentication schemes applied either in the application itself, such as ASP.net authentication, or applied in the host, such as IIS authentication. It is common practice for applications to maintain their own security domain.

Authorization

Application performs its own access control checks

Confidentiality

Application can apply an encryption scheme to some or all data fields

Integrity

Digital Signatures can be used for tamper detection. Hashes are used for some or all data fields.

4.2 Security Threats

Most threats today are targeted towards business applications. Web Services, as a software system[1], are therefore affected from application vulnerabilities and more especially web application vulnerabilities. This section will be divided into two parts. The first one will present generic web application vulnerabilities which affect Web Services Security. The second part will focus on vulnerabilities affecting directly Web Services standards and implementations. There will be presented some of the most well-known and popular Web Services attacks.

According to OWASP, the most critical Web Application Security Vulnerabilities for 2003 and 2004 were the following representing in figure 17.

[1] According to W3c definition

New Top Ten 2004	Top Ten 2003	New WAS Thesaurus
A1 Unvalidated Input	A1 Unvalidated Parameters	Input Validation
A2 Broken Access Control	A2 Broken Access Control (A9 Remote Administration Flaws)	Access Control
A3 Broken Authentication and Session Management	A3 Broken Account and Session Management	Authentication and Session Management
A4 Cross Site Scripting (XSS) Flaws	A4 Cross Site Scripting (XSS) Flaws	Input Validation->Cross site scripting
A5 Buffer Overflows	A5 Buffer Overflows	Buffer Overflows
A6 Injection Flaws	A6 Command Injection Flaws	Input Validation->Injection
A7 Improper Error Handling	A7 Error Handling Problems	Error Handling
A8 Insecure Storage	A8 Insecure Use of Cryptography	Data Protection
A9 Denial of Service	N/A	Availability
A10 Insecure Configuration Management	A10 Web and Application Server Misconfiguration	Application Configuration Management Infrastructure Configuration Management

Figure 17: OWASP Ten Most Critical Web Application Security Vulnerabilities 2004 update

Few years later, OWASP Top Ten Lists shows small changes on the ratings of risks affecting Web Applications:

OWASP Top 10 – 2007 (Previous)	OWASP Top 10 – 2010 (New)
A2 – Injection Flaws	A1 – Injection
A1 – Cross Site Scripting (XSS)	A2 – Cross-Site Scripting (XSS)
A7 – Broken Authentication and Session Management	A3 – Broken Authentication and Session Management
A4 – Insecure Direct Object Reference	A4 – Insecure Direct Object References
A5 – Cross Site Request Forgery (CSRF)	A5 – Cross-Site Request Forgery (CSRF)
<was T10 2004 A10 – Insecure Configuration Management>	A6 – Security Misconfiguration (NEW)
A8 – Insecure Cryptographic Storage	A7 – Insecure Cryptographic Storage
A10 – Failure to Restrict URL Access	A8 – Failure to Restrict URL Access
A9 – Insecure Communications	A9 – Insufficient Transport Layer Protection
<not in T10 2007>	A10 – Unvalidated Redirects and Forwards (NEW)
A3 – Malicious File Execution	<dropped from T10 2010>
A6 – Information Leakage and Improper Error Handling	<dropped from T10 2010>

Figure 18: OWASP Ten Most Critical Web Application Security Vulnerabilities 2010 update

Next, it follows a short analysis of risks with the top ratings which implicitly affect Web Services Security.

The following figure represents OWASP Top Ten 2010 Vulnerabilities examining the popularity and amenity of these attacks to happen as well as their detectability level and technical impacts.

RISK	Threat Agents	Attack Vectors	Security Weakness		Technical Impacts	Business Impacts
		Exploitability	Prevalence	Detectability	Impact	
A1-Injection		EASY	COMMON	AVERAGE	SEVERE	
A2-XSS		AVERAGE	VERY WIDESPREAD	EASY	MODERATE	
A3-Auth'n		AVERAGE	COMMON	AVERAGE	SEVERE	
A4-DOR		EASY	COMMON	EASY	MODERATE	
A5-CSRF		AVERAGE	WIDESPREAD	EASY	MODERATE	
A6-Config		EASY	COMMON	EASY	MODERATE	
A7-Crypto		DIFFICULT	UNCOMMON	DIFFICULT	SEVERE	
A8-URL Access		EASY	UNCOMMON	AVERAGE	MODERATE	
A9-Transport		DIFFICULT	COMMON	EASY	MODERATE	
A10-Redirects		AVERAGE	UNCOMMON	EASY	MODERATE	

Figure 19: OWASP Top Ten Risk Factor Summary 2010

The above attacks presented in figure 18 to 20, are common attacks in every Service Oriented Architecture. In the following analysis, I will mention how these attacks can happen and affect Web Services.

4.2.1 Common Web Application Attacks

Command Injection

It a result of the application accepting input from the browser that allows the attacker to insert commands and execute them. These attacks could range from database queries (SQL injection) to JavaScript (as in Xss) and the ability of inserting HTML into a page or even actual system commands.

i) SQL Injection:

Injection flaws, such as SQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing unauthorized data.

ii) Cross Site Scripting (XSS): XSS attack is injection of malicious content. It is based on the dynamic nature of the web sites and occurs whenever an application takes untrusted data and sends it to a web browser without proper validation and escaping. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.

Broken Authentication and Session Management

Often such elements as account credentials and session tokens are not adequately protected. This enables attackers to compromise such key elements as keys, passwords, and authentication tokens, thereby allowing for the assumption of user identities.
Application functions related to authentication and session management such as credentials and session tokens are often not implemented correctly, allowing attackers to compromise passwords, keys, session tokens, or exploit other implementation flaws to assume other users' identities. A typical attack example is Session spoofing or Hijacking.

Security Misconfiguration

Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform. All these settings should be defined, implemented, and maintained as many are not shipped with secure defaults. This includes keeping all software up to date, including all code libraries used by the application

Denial of Service Attacks (DOS) It is considered one of the most difficult problems faced by IT in general. This kind of attack showers the server computer with an array of requests leading the target machine out of order. Its goal, therefore, is the availability of the target blocking its resources from the legitimate users. This is a usual phenomenon in web servers / web applications. Although DOS attack may be part of a larger attacking strategy, it has two basic types. The first type exhausts the resources of the target making the system to consume its resources by itself. The second one blocks all communications disturbing interaction between users and the target system. In the case of a Web Service, a DOS attack can exhaust infrastructure's resources (application, server, network) where the service depends on or disrupt their configuration (for example routing information).

Buffer overflows

Buffer overflows is considered a very popular method of attack. It overloads the web server with payloads containing excess information. Usually this information contains parameters that are longer than the Web Service expects. Thus, this information overflows the server resulting in a system crash. The usual way of executing buffer overflows is through internet forms that web sites use in order to collect information. Hackers implement certain alterations in the HTML source code bypassing the limitation of characters that forms use for each field. An application which is not able to manage overflows of input in its forms, the time it receives the altered lengthy forms it is not prepared to respond. In an effort to cope with this new situation, the application overflows parts of its memory resulting in a system malfunction or crash.

The attacks presented are the most recurrent and critical attacks happening in web applications. However, this is only a small portion of the whole picture of

Table 1 illustrates a summary of further known attacks no matter of their frequency and ratings as web applications attacks.

Cookie poisoning	Message alteration	Man in the middle	Principal spoofing	Forged claims	Replay of message parts
Url access restriction failures	Forceful browsing	Injection flaws	Improper error handling	Insecure storage	Insecure configuration management
AJAX security	Falsified messages	*Cross Site Request Forgery (CSRF)*	Insecure communications	Information leakage	Malicious file execution
Brute force	*Dictionary attacks*	Response splitting	Backdoor and debug options	Sophisticated http attacks	Stealth commanding

Table 1: some more web application attacks that indirectly affect Web Services

4.2.2 XML-based attacks

Unlike the usual methods of attacking the system and penetrating it, followed by exploiting the faults in the operating system (for example trying to get a password) or the web application, attackers can use an ordinary SOAP or XML request in order, for instance, to delete crucial data, and add or recover any confidential information on the web server. Additionally, they can carry out other malicious tasks by posing threats through the published Web Service. Vulnerabilities concerning XML/SOAP technologies are actually affecting Web Services in the most direct way. In this section will be presented and analyzed some well-known XML-based attacks affecting mostly the Message Layer Security.

A graphic representation of attacks will provide the reader with better understanding of where each attack on SOAP message takes place.

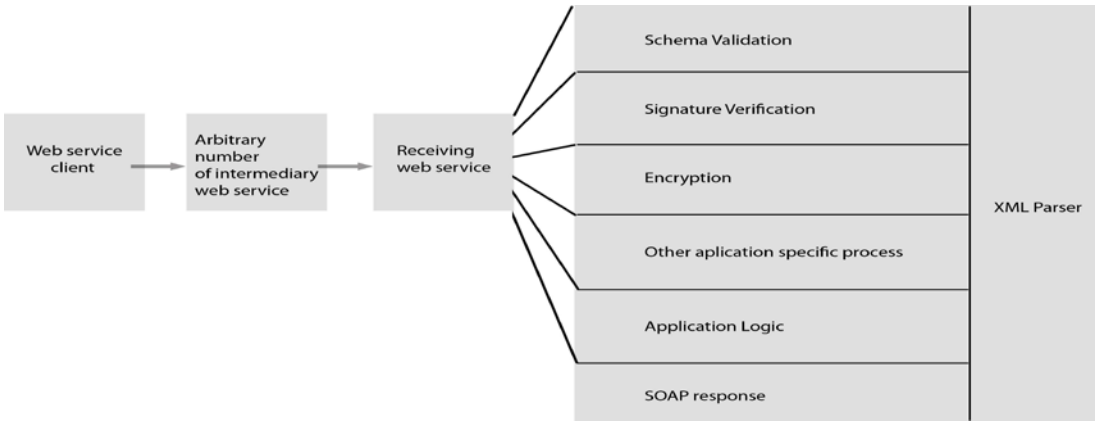


Figure 20: Graphic representation of Web Services XML-based attacks

XML Injection

The attack aims at the XML Parser.

It is a type of injection attack where targets the XML tags in the SOAP message in order to modify the SOAP structure. A typical result of such attack is the execution of a restricted operation.

XPath Injection

The attack aims at the application logic performing queries not intended by the developer.

XPath is similar to SQL injection attack but more dangerous. XPath as a query language, often used with C# and JavaScript, uses parameters from an XML document to create an XPath query. In case users input is not validate, XPath query can be modified according to the attacker's will. As a result, the attacker could be read even the entire XML document considering the query it was used.

SOAPAction Spoofing

Violating Authorization (Access Control violation)

The attack aims at the application logic in order to execute a prohibited operation.

SOAPAction is an optional HTTP Header which is used only to inform the Web Service what operations are contained in the SOAP body. The vulnerability exists on certain Web Service frameworks that determine the operation to be executed solely on the information contained in the SOAPAction attribute.

WSDL Disclosure

Violating Confidentiality

WSDL Disclosure aims at discovering WSDL files hidden from the public. The attack aims at disclosing the metadata of the web service and not any special web service component. Depending on the discovered web service various further attacks are possible.

XML Flooding

Limiting Availability

Similar to a DOS attack, the target is the web server and not a specific service component. The attacker floods the web service with numerous legitimate SOAP messages.

XML Denial of Service (XDOS):

Limiting Availability

Similar to DOS attacks, XDOS aim at exhausting the resources that the Web Service uses by sending large XML messages. XDOS could target only a part of an XML document such as the Header or the body or place oversize content in SOAP envelope in general. Furthermore,

XDOS could attack an XML tag exploiting for example the length of the element name (oversized XML attack). There are many sub-types of XDOS attacks. For example, XML document size or XML oversize payload attack sends very large SOAP messages to the XML parser. XML Encryption and XML Signature Transformation DOS attacks create large overlaps during the encryption or signature creation process in a SOAP message. Otherwise XML Signature Transformation DOS could use an array of signatures attached to an XML message received by the target where a naïve XML parser would try to examine each signature thus exhausting its resources. Another type of XDOS is also attack obfuscation.

4.3 Pen Testing Web Services

It is outside the scope of this thesis to examine penetration testing as industry standard concerning the existing methodologies, techniques and processes. Penetration testing is a special sector of Information Security industry with great importance which has, finally, left the labs and controlled environments and it is running on the real life. Penetration testing differentiates from the classic hacking or cracking and it is self-defined as Ethical Hacking or White Hat Hacking. Besides, the final purpose of penetration testing is to evaluate and improve security.

For the purpose of my own Web Service implementation I decided to use an oversimplified penetration testing model based on the Zero Entry Hacking Penetration (ZEH) Testing Methodology. This model, I call it “Wcf zero entry model”, is my guide of assessing my Wcf Service Security providing a proof of concept (POC) that vulnerabilities are real. According to this model, I assume I have no knowledge (“zero entry” concept) of the target prior to penetration testing. Therefore, at first, I have to collect data, then find possible vulnerabilities and finally exploit these vulnerabilities.

The “Wcf zero entry” model consists of 4 phases:

A) Phase one: Information Gathering

This phase targets at getting as much information as possible about the target. Such information could be a list of domains and IP address, a list of emails that could be used for later authentication login attempts and any other piece of information that accidentally resides on the internet and could be used against the target.

B) Phase two: Scanning

It is divided into (i) port scanning and (ii) vulnerability scanning. Port scanning is providing a list of open ports and potential services running on these ports while vulnerability scanning is locating and identifying weaknesses on software and services of the target.

C) Phase three: Exploitation

The goal here is to use any information gathered in the previous two phases and conduct the real attacks. The final result should be getting administrative access over the target’s system.

D) Phase four: Reporting

This is the last phase where it is created and presented the final report containing a summary of the processes being used but also the findings of that processes. This final report can be a measure against other penetration testing models and reports of evaluating the security of the Wcf Service.

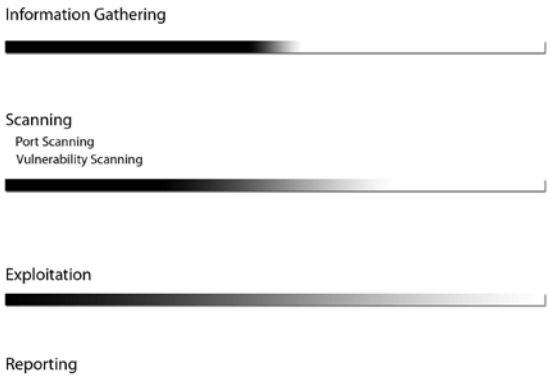


Figure 21: The "Wcf Zero Entry" penetration testing model and its phases

4.3.1 Case Study: Wcf Service Vulnerability Assessment

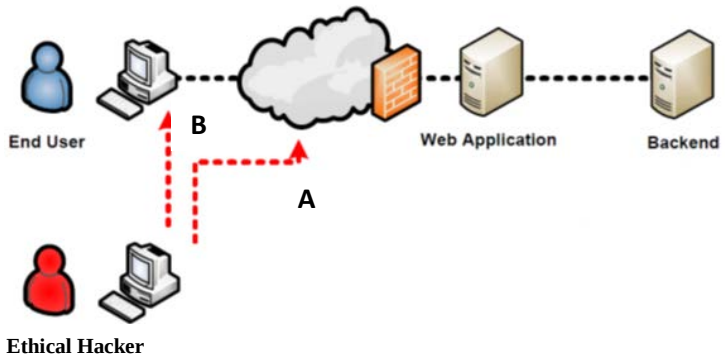


Figure 22: The attack scenario

The attack scenario includes a real working system behind a corporate firewall which is property of Company x; the details of the Company will be tactfully filtered (for example ip addresses, domain names etc.). I play the role of the attacker as an ethical hacker (White Hat) who will try some penetration testing (from now on called pentest) techniques having zero knowledge of the target. The only knowledge I have as a hacker is the name of Company x and therefore the only point of attack is route A as shown in figure 23. In this point, an assumption will make the attack scenario more interesting. Supposedly, one of my neighbors (end user) is accessing the service from her computer and home network. Then it is automatically created a second point of attack, route B, targeting client's machine (service consumer) as shown in figure 23. The final purpose of this attack scenario is to locate the Wcf Service and assess its security. As a result, neither penetration testing phases and attack processes are going to be fully analyzed, nor the final outcomes of these processes.

Next table illustrates the ethical hacker's toolkit used during the penetration testing phases.

Tools	OS	Category	Type
HTTRACK	Linux – Backtrack 5	Information Gathering	Offline Browser Utility
NetCraft	Linux – Backtrack 5	Information Gathering	Information Gathering Utility
Netcat, Xprobe2	Linux – Backtrack 5	Information Gathering	Operating System Fingerprinting
The Harvester	Linux – Backtrack 5	Information Gathering	Information Gathering Utility
Fping	Linux – Backtrack 5	Scanning	Ping Hosts Utility
Nmap – Zenmap	Linux – Backtrack 5	Scanning	Network Discovery and Security auditing
Nessus	Linux – Backtrack 5	Scanning	Vulnerability and Configuration assessment
Fiddler	Windows 7	Exploitation	Web Debugger
ZAP proxy, Web Scarab	Windows 7	Exploitation	Integrated Pentesting tool
W3af	Windows 7	Exploitation	Web Application Attack and Audit Framework
Medusa, John the Ripper	Linux – Backtrack	Exploitation	Login Brute Forcer
Metasploit	Linux – Backtrack 5	Exploitation	Integrated Pentesting tool
Wireshark	Linux – Backtrack 5	Exploitation	Network analyzer
Dsniff	Linux – Backtrack 5	Exploitation	Network Sniffing - MITM
Ettercap	Linux – Backtrack 5	Exploitation	Network Sniffing - MITM
Nikto	Linux – Backtrack 5	Exploitation	Web Vulnerability Scanner
WebSecurify	Linux – Backtrack 5	Exploitation	Web Vulnerability Scanner
SQLmap	Linux – Backtrack 5	Exploitation	SQL injections
Burp Suite	Windows 7	Exploitation	Integrated Pentesting tool
SoapUI	Windows 7	Exploitation	Testing Soap Message
SOA Cleaner	Windows 7	Exploitation	Testing Soap Message
WCF Storm	Windows 7	Exploitation	Testing Soap Message
Altova XMLSpy	Windows 7	Exploitation	Testing Soap Message

Table 2: Penetration Testing tools used, illustrated per category and operating system

Before analyzing the penetration testing process I have followed for the Wcf Service, I have to present my penetration checklist similar to OWASP Web Application Penetration Checklist. The following table illustrates the attacks I have tried to recreate; the exact name of the attack and the possible vulnerabilities these attacks exploit. This checklist is only a guide to assist me in the penetration testing processes and especially Phase 3. It can be considered as a sample of potential attacks applicable to the most of penetration testing processes.

Type of Attack	**Name of Attack**	**Exploits / Check if…**	**Notes**
1. Error Handling	Application Error Messages	Application presents application error messages to an attacker that could be used in an attack	
2. DOS	Application Flooding	Application will stop working correctly in case of large volumes of requests, transactions, and/or network traffic.	Use various fuzzing tools to perform this test (e.g., SPIKE)
3. Access Control	i. Parameter Analysis ii. Authorization Parameter Manipulation	There is no access control model enforced by the application to secure parameters tampering It is possible to change the session ID's parameter after a valid user has logged in	Typically, this includes manipulation of form fields, URL query strings, client-side script values and cookies. .eg username
4. Authentication	i. HTTPS for authentication endpoint request	SSL is not used for end-to end point protection	Certification needs to be provided from the server
	ii. Authentication bypass	Authentication process can be bypassed	Typically, this happens in conjunction with flaws such as SQL Injection.
4.b Session Management	Session Token Length	Adequate length of session token	Provide protection from guessing (dictionary attacks) during an authenticated session
5. Configuration Management	i. HTTP Methods	The Web server does not support the ability to manipulate resources from the Internet (e.g., PUT and DELETE).	
	ii. Application Admin Interface	Application administrative interface accessible to the Internet	
	iii. Known Vulnerabilities / Security Patches	There are presented known vulnerabilities that vendors have not patched	

6. Data Protection	Data integrity	This typically occurs when an attacker can tamper data parameters.	
7. Input Validation	i. SQL Injection	Ensure the application will not process SQL commands from the user.	
	ii. Cross Site Scripting	Ensure that the application will not store or reflect malicious script code.	

Table 3: Penetration checklist

Analyzing the penetration testing phases

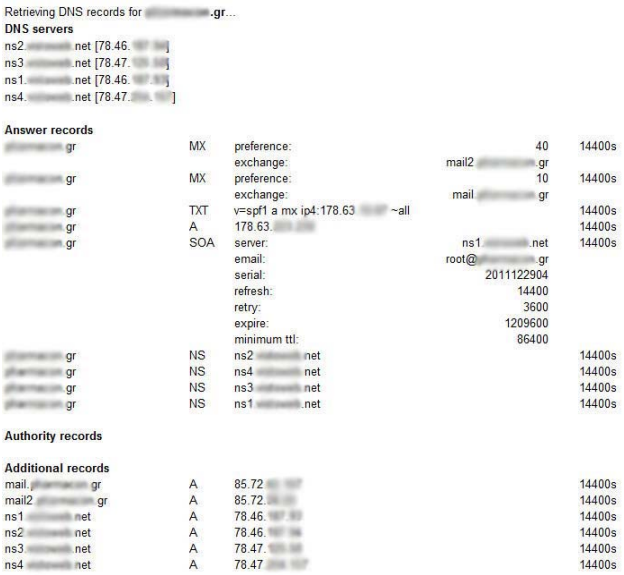
Phase 1: Information Gathering

Proceeding to the first phase of the pentest, my initial purpose is to start gathering what is in the public eye about company x. The only available information is the company's name, so making a search on the internet I easily find the site of the company. I use HTTRACK to copy the entire web site, page-by-page in a local folder for accessing it offline, thus minimizing my exposure (digital fingerprints) to systems admins. By examining the site and its code but also conducting web whois (domain name check), I realize that the site is outsourced or managed by a third party.

Figure 23: Web Whois at: https://grweb.ics.forth.gr/Whois?lang=en

In addition, I compare site's information with Google's cache in case there are changes between the two site's versions (type in Google search: "cache:domain name of company x "

Next, I try NS lookup (querying the DNS for retrieving resource records) for the domain x using an online tool at: http://network-tools.com . A similar tool for extracting information from DNS is called Dig.



Retrieving DNS records for [redacted].gr...

DNS servers

ns2.[redacted].net [78.46.[redacted]]
ns3.[redacted].net [78.47.[redacted]]
ns1.[redacted].net [78.46.[redacted]]
ns4.[redacted].net [78.47.[redacted]]

Answer records

[redacted].gr	MX	preference:	40	14400s
		exchange:	mail2.[redacted].gr	
[redacted].gr	MX	preference:	10	14400s
		exchange:	mail.[redacted].gr	
[redacted].gr	TXT	v=spf1 a mx ip4:178.63.[redacted] ~all		14400s
[redacted].gr	A	178.63.[redacted]		14400s
[redacted].gr	SOA	server:	ns1.[redacted].net	14400s
		email:	root@[redacted].gr	
		serial:	2011122904	
		refresh:	14400	
		retry:	3600	
		expire:	1209600	
		minimum ttl:	86400	
[redacted].gr	NS	ns2.[redacted].net		14400s
[redacted].gr	NS	ns4.[redacted].net		14400s
[redacted].gr	NS	ns3.[redacted].net		14400s
[redacted].gr	NS	ns1.[redacted].net		14400s

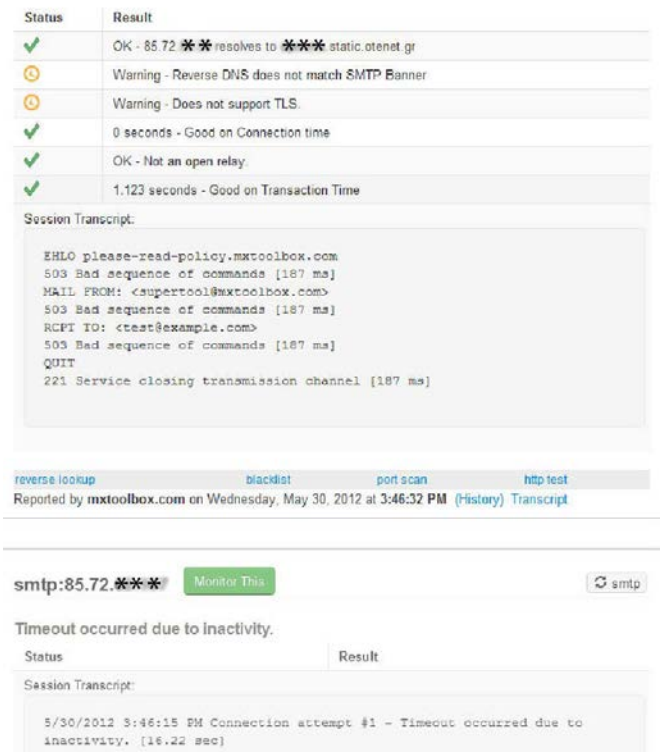
Authority records

Additional records

mail.[redacted].gr	A	85.72.[redacted]	14400s
mail2.[redacted].gr	A	85.72.[redacted]	14400s
ns1.[redacted].net	A	78.46.[redacted]	14400s
ns2.[redacted].net	A	78.46.[redacted]	14400s
ns3.[redacted].net	A	78.47.[redacted]	14400s
ns4.[redacted].net	A	78.47.[redacted]	14400s

Figure 24: NS Lookup

Another online tool: mxtoolbox.com can assist me more in querying the domain name providing mxlookup, smtp tests and even port scans, thus providing useful information for the second phase.



Status	Result
✔	OK - 85.72.**.** resolves to ***.static.otenet.gr
⏲	Warning - Reverse DNS does not match SMTP Banner
⏲	Warning - Does not support TLS.
✔	0 seconds - Good on Connection time
✔	OK - Not an open relay.
✔	1.123 seconds - Good on Transaction Time

Session Transcript:

```
EHLO please-read-policy.mxtoolbox.com
503 Bad sequence of commands [187 ms]
MAIL FROM: <supertool@mxtoolbox.com>
503 Bad sequence of commands [187 ms]
RCPT TO: <test@example.com>
503 Bad sequence of commands [187 ms]
QUIT
221 Service closing transmission channel [187 ms]
```

reverse lookup | blacklist | port scan | http test

Reported by **mxtoolbox.com** on Wednesday, May 30, 2012 at **3:46:32 PM** (History) Transcript

smtp:85.72.*** Monitor This | smtp

Timeout occurred due to inactivity.

Status	Result

Session Transcript:

```
5/30/2012 3:46:15 PM Connection attempt #1 - Timeout occurred due to
inactivity. [16.22 sec]
```

Figure 25: SMTP test Example

A very interesting tool for information gathering is called *the "Harvester"*[1]. It catalogs e-mail addresses, subdomains and hosts that are directly related to the target.

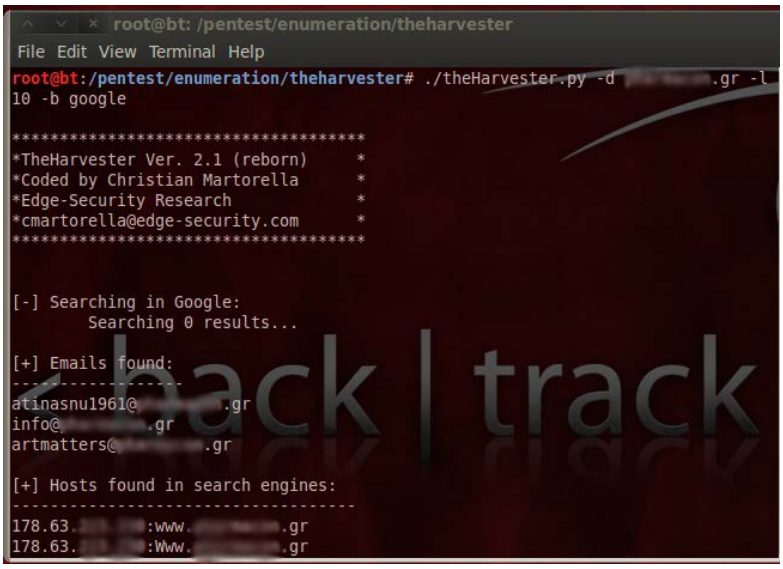


Figure 26: the Harvester results

Last but not least, Netcraft[2] is one network analysis online tool which assists in determining what a site is running!! Netcraft, as well as similar tools such as Httprint, Netcat and Xprobe2, conducts OS fingerprinting in order to reveal the version and type of a running web server, thus providing to the pen tester with more certain information to focus on.

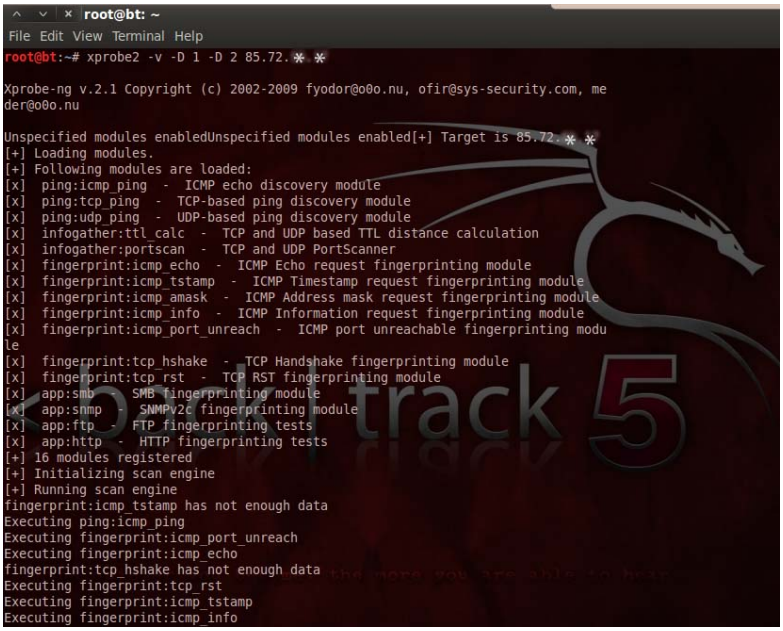


Figure 27: Xprobe2

[1] Shown in figure 27

[2] Available at http://news.netcraft.com/

Phase 2: Scanning

In the very first part of scanning phase, the scope is to determine whether the target is active (alive) or not. This can be done by ping tests. "*FPing*", is a simple tool of running pings. It includes scanning of hosts between a network range or just standalone hosts. However, it is not always possible to receive ping responses since many servers or firewalls may block ping packets.

Figure 28: pinging IP range

Whether or not there are successful results from ping sweeps, it is wise to proceed in port scanning of the system in order to locate specific ports and services hosted by the target machine. Usually, there are certain ports that allow certain services such as port 21 for FTP, port 23 for telnet, 443 for HTTP and many more. However, system administrators may not use standard ports for running services or may, again, set firewall restrictions. In addition, systems often filter their ports in order to conceal the services behind those ports. "Nmap"[1] is an excellent tool for port scanning. It includes a wide variety of scanning options (named switches) considering both TCP and UDP ports.

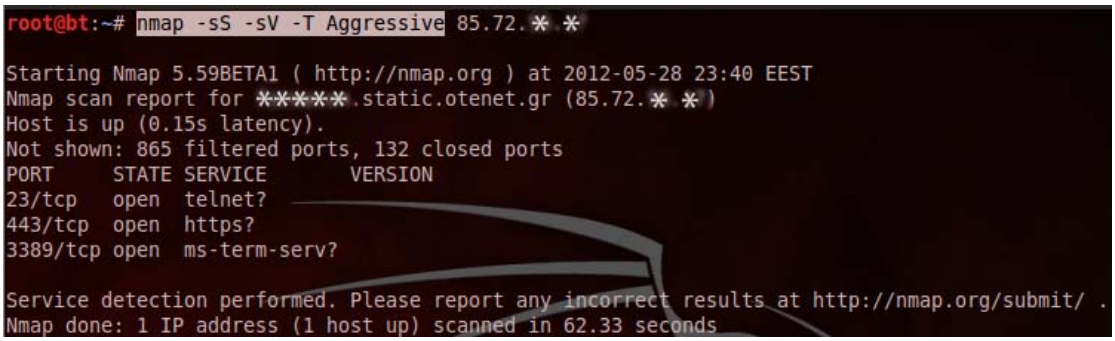


Figure 29: a SYN scan on the first 1000 ports using Nmap

Zenmap, as the official Nmap scanner GUI, provides a more easy way of using Nmap and presenting the results.

[1] Nmap is available at www.insecure.com

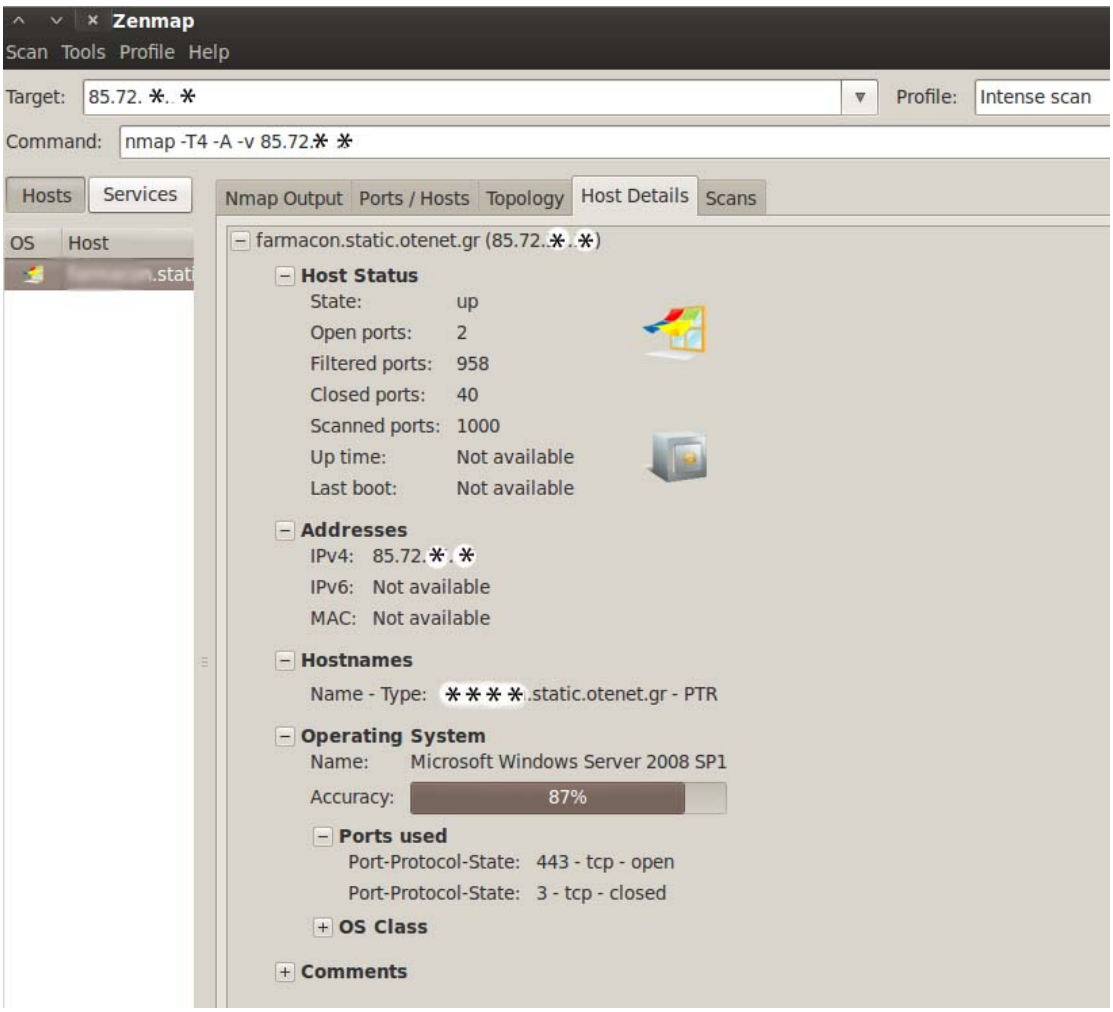


Figure 30: Zenmap GUI – Aggressive TCP scan.

It is not rare, system or software administrators to leave backdoors or allow remote access services in order to provide technical support. Attempting to log into those remote services (such as telnet, ssh, ftp or rdp) regardless the results of scanners, by using defaults or pieces of information gathered during the first phase of the penetration testing, can lead to direct security breaks without the need for conducting further exploits. “Telnet” is a typical example of remote service that very often offers a potential break point for an attacker.

Figure 31: Connecting to a remote service

After discovering ports and services, it is time for the second part of scanning phase which includes vulnerability scanning. Vulnerability scanners, such as the famous Nessus, target unpatched systems and services in order to exploit known vulnerabilities like remote code execution.

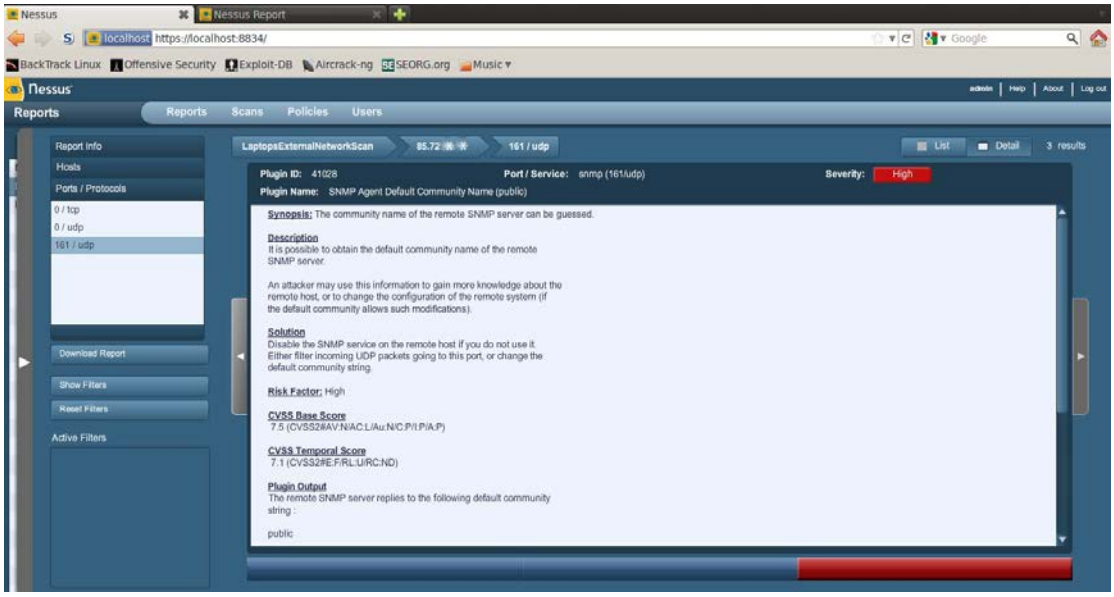


Figure 32: Nessus report synopsis of certain vulnerability for service snmp.

Nessus provides Plugin IDs for integration with network exploitation tools and especially the Metasploit Framework in order to proceed into more specific attacks in the exploitation phase. Nexpose is another vulnerability scanner integrated with Metasploit Framework.

Phase 3: Exploitation

This is probably the most interesting phase in a penetration test. The end goal here is to gain administrative-level access to the victim's machine in order to alter the original functionality of the system. The exploitation phase is a free-flowing phase because of the uniqueness of each target. As a result different tools are required to be used according to the operating system, software and services of the end target. In this point I must mention the fact that although each one of the numerous existing tools has a distinct operation (e.g fuzzing, sniffing etc.) in many cases the same tools can be used to conduct more than one types of attacks.

To begin with, Network Sniffing – hijacking is a very important exploitation stage. Programs such as Dsniff, Ethereal, Ettercap, Metasploit[1] and Wireshark which conduct thorough network - protocol analysis, can provide with excellent network sniffing results. These tools provide the appropriate environment for passive and active session hijacking attacks. An

[1] Metasploit, is an excellent integrated penetration testing tool, using many build-on modules such as Armitage, Nexpose and Fast-track

example of passive hijacking attack is session replay where the attacker (MITM) captures packets and modifies the data before sending it to the target. In our attack scenario, network sniffing applies to Route B and can provide information about the exact location of the Web Service, the name and port as well as credentials for accessing the service. Starting sniffing the network traffic of end user, I use Wireshark to capture all packets. By watching the traffic I notice the use of IP address (85.72.**.**) and port 82.

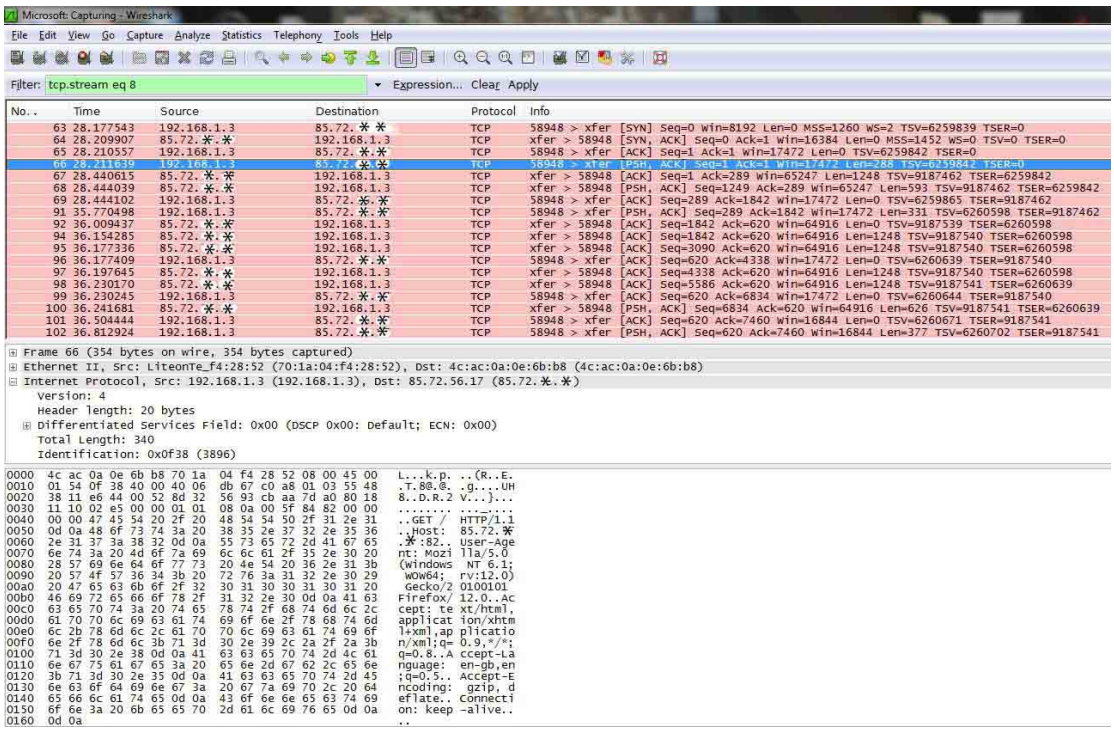


Figure 33: Network analysis using Wireshark

The end user uses credentials to log in to the server. Ettercap, Armitage (a metasploit module) or Dsniff are password sniffing and network traffic analysis tools that can be used to capture the sensitive data of user credentials. Figure 34 illustrates the successful results of sniffing username and password.

Figure 34: Sniffing user credentials with Dsniff

As we can notice, this is a very easy process in case the server uses basic authentication, because password is communicated over the network in clear text. Basic authentication provides no security and therefore is rare in use unless it is used in conjunction with other security mechanism such as SSL. The server could use a more secure authentication mechanism such as Integrated Windows Authentication. Nevertheless, the password sniffing is still possible although it is now a more painful process for an attacker.

Microsoft uses session IDs to process requests from the users. The most commonly used method delivering session IDs is by using cookies sent to the user/s browser. The session ID, a 120-bit random number, remains the same until the user browses outside the DNS domain or closes the browser. The cookie is not persistent and has expiration time which is set by default in 30 minutes. However, an attacker in this time window can still steal or poison the cookie, thus gaining access to the privileges of end user. This is a classic session hijacking attack where the stolen Session Id can be used. Apart from that, dedicated password cracking tools or brute forcers such as medusa, Cain & Abel, and john the ripper can crack even the LM – NTML hash that ASP.NET uses to protect its session Id.

Request Header Name	Request Header Value
Host	85.72.*.*:82
User-Agent	Mozilla/5.0 (Windows NT 6.1; WOW64; rv:12.0) Gecko/20100101 Firefox/12.0
Accept	*/*
Accept-Language	en-gb,en;q=0.5
Accept-Encoding	gzip, deflate
Proxy-Connection	keep-alive
Content-Type	application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With	XMLHttpRequest
Referer	http://85.72.*.*:82/
Content-Length	48
Cookie	ASP.NET_SessionId=sjcscqy0bkei2vtsyi1a0df5
Pragma	no-cache
Cache-Control	no-cache
POSTDATA	modelID=SELECT * FROM characteristics&actionID=2

Response Header Name	Response Header Value
Status	Unauthorized - 401
Content-Length	1656
Content-Type	text/html
Server	Microsoft-IIS/6.0
WWW-Authenticate	NegotiateNTLM
X-Powered-By	ASP.NET
Date	Wed, 23 May 2012 16:33:13 GMT
Proxy-Support	Session-Based-Authentication

Figure 35: ASP.NET Session Id

Moving to next attacks, Zed Attack[1] Proxy is such a tool, considered as a" Swiss jack-knife" for testing web applications and Web Services. It provides several testing techniques such as active scanning, spiders[2], brute forcing, port scanning and fuzzing[3]. Starting to examining my penetration checklist, I use ZAP to test my Wcf Service for input validation vulnerabilities. More specifically I use ZAP as fuzzer testing Cross Site Scripting vulnerability as shown in figure 36.

[1] Provided by OWASP

[2] Browsing and capturing resources related to the web service (mostly used in phase one)

[3] Providing invalid, unexpected, or random data to the inputs of the service

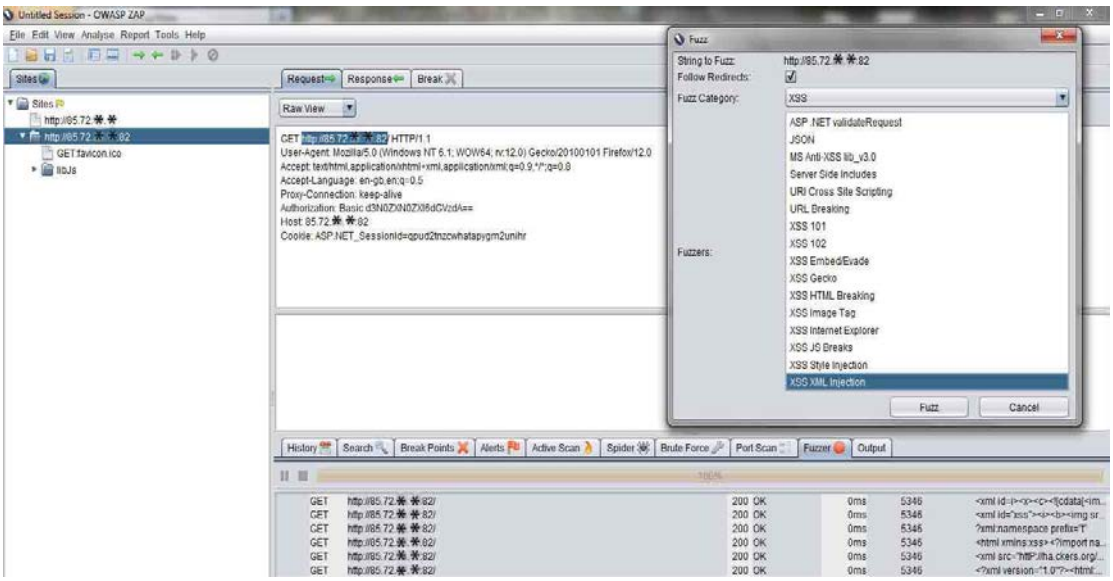


Figure 36: Fuzzing (OWASP ZAP)

In this point it is useful to juxtapose another script injection attack similar to XSS. This is the well-known SQL injection attack. This attack can run either manually by using scripts and logic true commands (such as 'or'1'='') or automatically. Sqlmap is an automatic SQL injection and database takeover tool.

```
root@bt:/pentest/database/sqlmap# ./sqlmap.py -u http://85.72.*.*:82

    sqlmap/1.0-dev (r4009) - automatic SQL injection and database takeover tool
    http://sqlmap.sourceforge.net

[!] Legal Disclaimer: usage of sqlmap for attacking web servers without prior mu
tual consent can be considered as an illegal activity. it is the final user's re
sponsibility to obey all applicable local, state and federal laws. authors assum
e no liability and are not responsible for any misuse or damage caused by this p
rogram.

[*] starting at: 01:45:06

[01:45:07] [INFO] using '/pentest/database/sqlmap/output/85.72.*.*/session' as
 session file
[01:45:07] [INFO] testing connection to the target url
[01:45:08] [CRITICAL] not authorized, try to provide right HTTP authentication t
ype and valid credentials (401)
[01:45:08] [WARNING] HTTP error codes detected during testing:
401 (Unauthorized) - 1 times

[*] shutting down at: 01:45:08
```

Figure 37: Automated SQL injection attack using Sqlmap

Tamper Data

This is a testing technique aiming mostly at the integrity of the Web Service. The service does not provide any security at the message layer, thus enabling the attacker to change the parameter values. This is considered a MITM attack mentioned in network sniffing.

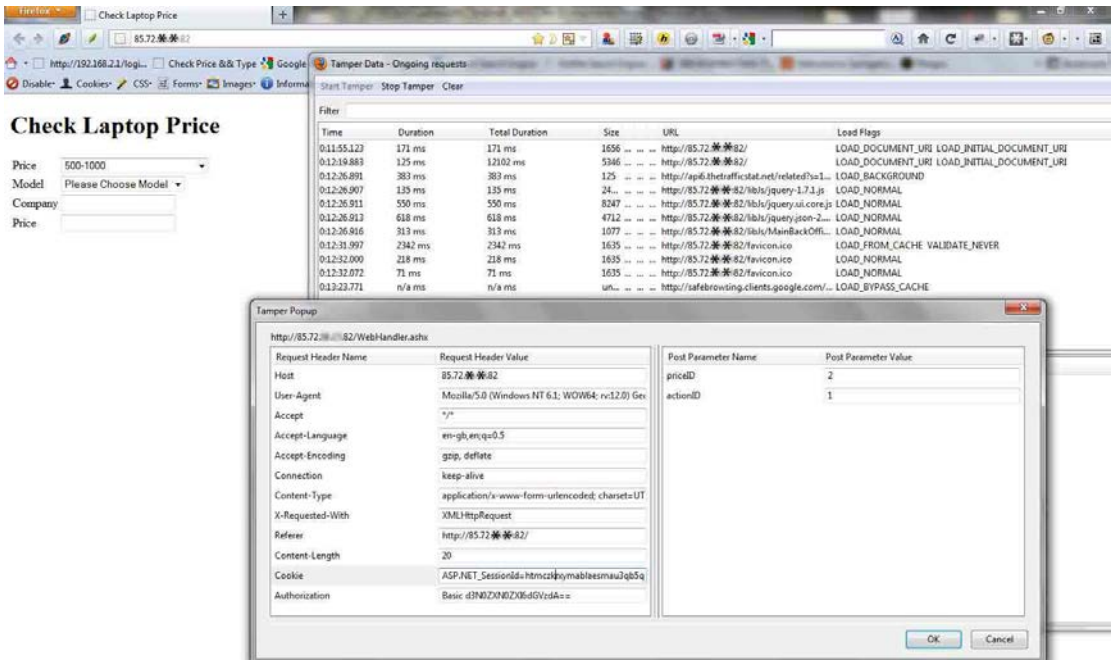


Figure 38: Tampering Data

Fiddler is a Web Debugging Proxy which logs all HTTP(S) traffic between your computer and the Internet

Error Handling

Detailed error messages are turned on. These detailed errors leak important information that may be used to stage further attacks on the web server.

Figure 39: Error message Directory Listing

The above figure illustrates the information that an attacker can obtain about the existing virtual directory on this specific URI. Next figure, illustrates another message error in an unsuccessful authorization attempted providing at first glance information about the Web Server and certain message (401.2) about its configuration. Although this is valuable information for system administrator, it can also be valuable information for an attacker.

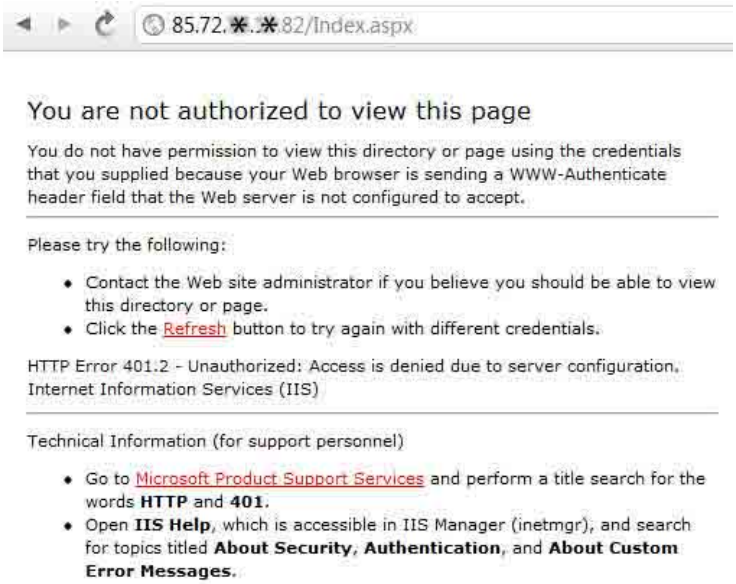


Figure 40: Authorization Error Message 401.2

Web-based Exploitations

Nikto is a web server scanner which performs comprehensive tests against web servers for multiple items.

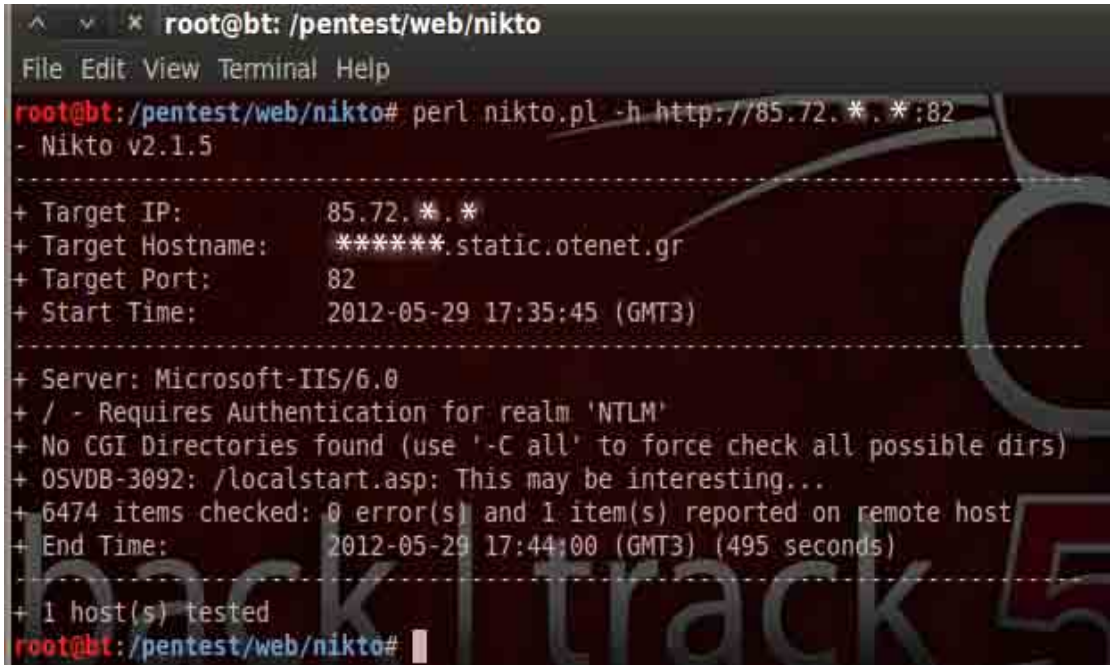


Figure 41: Nikto web vulnerability scanner

WebSecurify

WebSecurify is an advanced testing solution built to quickly and accurately identify web application security issues. It is very fast, simple and convenient program in use which automates the web application security vulnerability assessment.

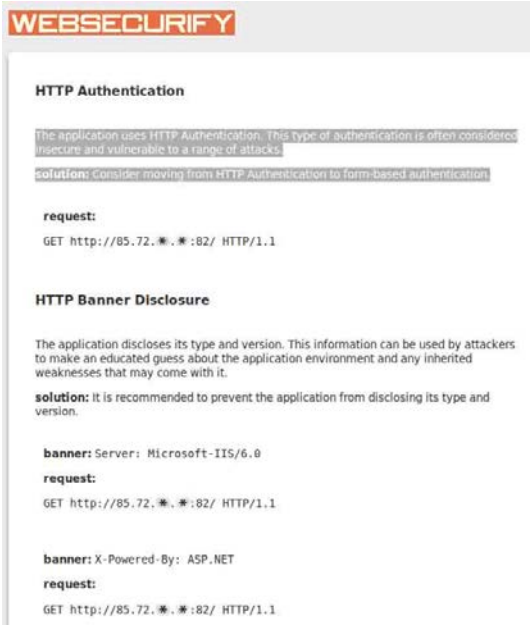


Figure 42: WebSecurify testing solution

Last but not least, it is Burp Suite, one of the best integrated platforms for performing security testing of web applications. Burp combines advanced manual techniques with state-of-the-art automation making attacks faster and more effective. It includes various tools such interception proxy, spider, scanner, intruder etc.

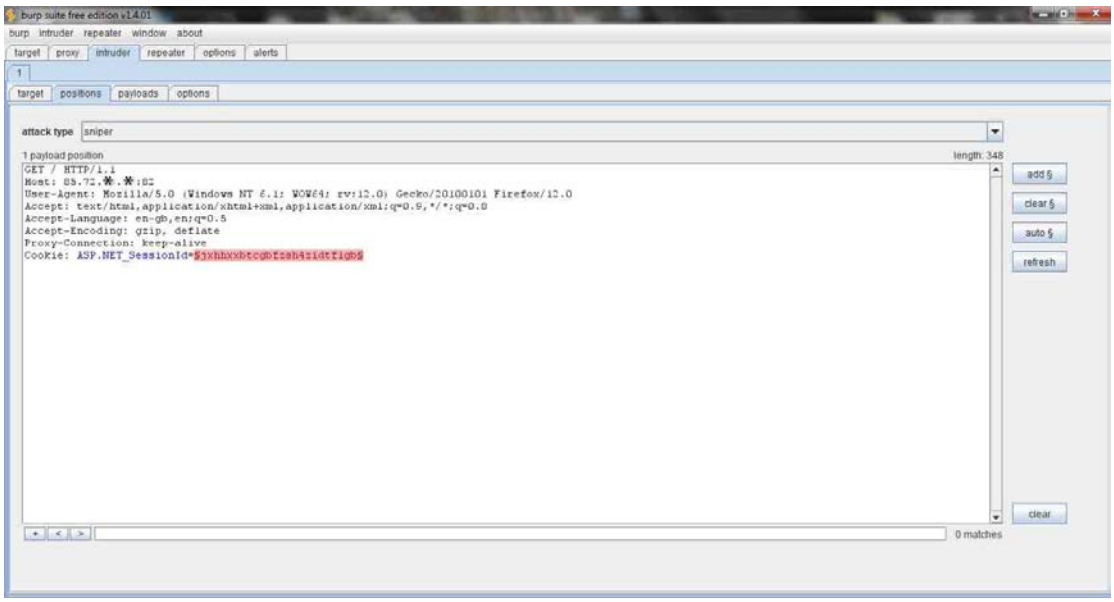


Figure 43: Burp Suite

Phase 4: Final report

This is the final phase writing the penetration testing report. This phase summarizes the processes and findings of penetration testing as well as the final outcomes.

Phase one includes information gathering about the target. Any piece of information can be valuable, guiding precisely next phases and utilizing better the existing attacking resources. The more knowledge of the target, the better. The goal of phase one is o end-up with IP addresses of hosts on the target network, open UDP/TCP ports and types of operating systems.

Phase two provides with the scanning tools for identifying the characteristics of the host and the possible vulnerabilities preparing the environment for phase three to exploit the findings.

Phase three can be endless. The most important element of this phase is the time limit. All ethical hackers have to stick to a tight time plan where they have to run the attacks. In addition, time is most critical for some attacks than others. For instance, stealing the cookie in session hijacking must succeed in a very limited time-frame. In general, this is a very free-flow phase with numerous testing solutions. Thus, time and effort should be carefully balanced. Good organization of the previous two phases will be critical to the success of this phase.

Table 4 summarizes the attack results according to the penetration checklist provided at the begging of the penetration testing section. The column “Assessment Tools” indicates only some of the assessment tools used to complete the attacks.

Type of Attack	Name of Attack	Indicative tools used	Successful Attack
1. Error Handling	Application Error Messages	Fiddler	YES
2. DOS	Application Flooding	SoapUI, Nmap	YES
3. Access Control	i. Parameter Analysis ii. Authorization Parameter Manipulation	Tamper Data, SoapUI	YES YES
4. Authentication	i. HTTPS for authentication endpoint request	Fiddler	YES
	ii. Authentication bypass	Fiddler	NO
4.b Session Management	Session Token Length	John the Ripper, Ettercap	NO
5. Configuration Management	i. HTTP Methods	HttpRequester	NO

	ii. Application Admin Interface	Fiddler	NO
	iii. Known Vulnerabilities / Security Patches	Nessus	YES
6. Data Protection	Data integrity	Tamper Data SoapUI	YES
7. Input Validation	i. SQL Injection	Sqlmap	NO
	ii. Cross Site Scripting	ZAP	NO

Table 4: Attack Results according to penetration checklist

4.3.2 Proposed Countermeasures

There are two types of security you can define in WCF: transport level and message level.

Transport Layer Security

Secure Socket Layer (SSL). SSL is the default protection mechanism for Web Services being in transit. It allows multiple users to access the service protecting the confidentiality.

It is very easy to enable SSL in IIS. Before doing so, it is necessary to create a trust certificate for users who will access the service through SSL. Certificates can be requested from trusted third parties or can be created self-signed certificates. In our case, I will use a self-signed certificates named “Laptops”.

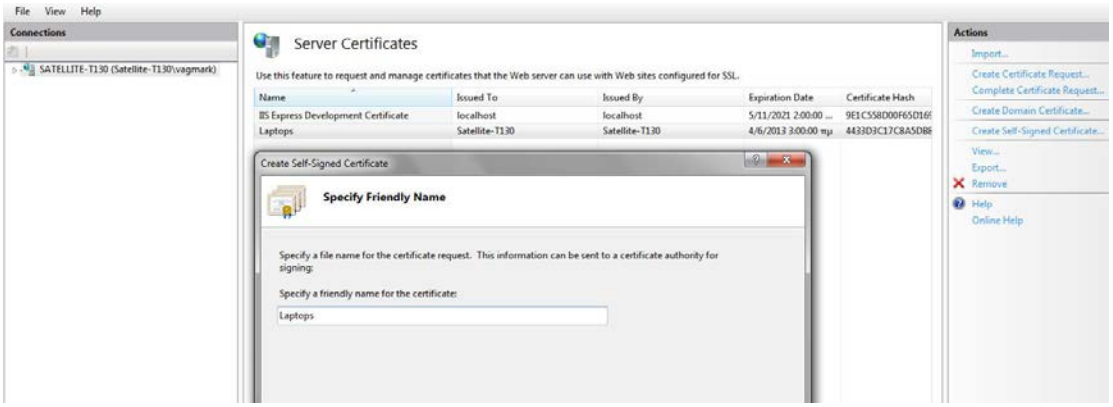


Figure 44: Creating a Self-signed SSL certificate

As soon as the certificate is created, the next step is to enable SSL for the web site (application) which hosts the Web Service named SecureLaptops.

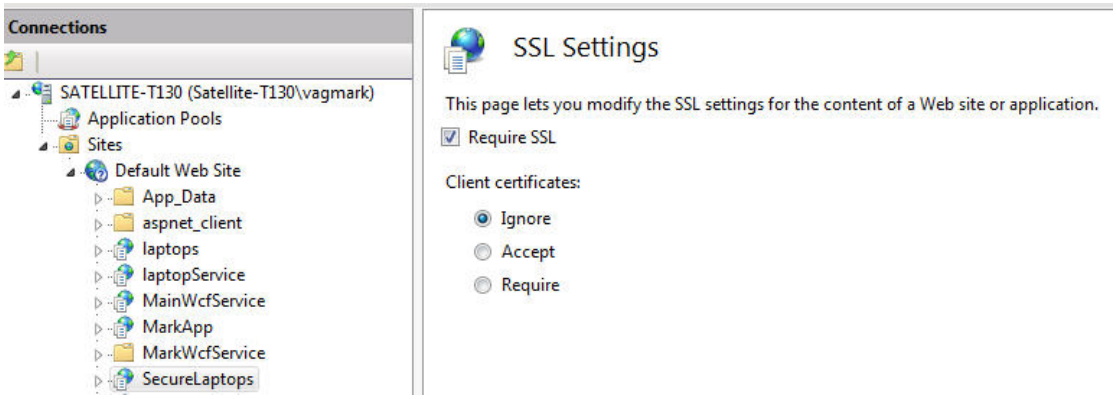


Figure 45: Enabling SSL in IIS

With SSL enabled, when a self-signed certificate is used, the end-users are prompt to trust the site they visit or not. As an extra security feature, I could set mutual authentication of client and server to prevent non-repudiation and Man-in-the-Middle attacks. In that case Web Server should be configured to require client certificates. In this case, it is better to use a PKI infrastructure provided by a trusted third-party.

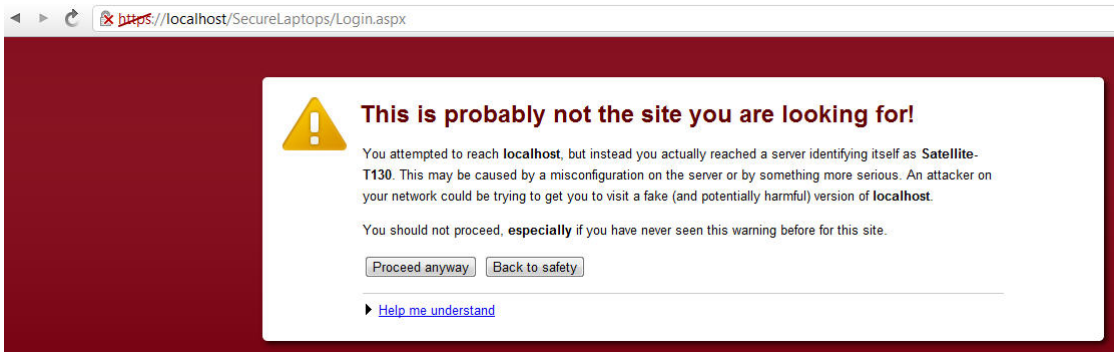


Figure 46: Security warning when visiting a site with self-signed SSL certificate.

SSL is a basic option for end-to-end point encryption. However, it can only protect the data only between two end points while multiple endpoints are not addressed. As a result, security of the message itself is very important and will be examined in the next section.

Virtual Private Networks (VPNs). VPN technology is an alternative to SSL Transport Layer Security Technology. I could use a virtual private network to create secure communications tunnels with the service consumers. The type of VPN could be either site-to-site VPN for more static connections or running an SSL-VPN client in end-users' machines. VPNs, however, are not considered a good solution for large networks that serve huge number of remote users.

Bindings

This is probably the most important feature that differentiates the structure of my Wcf Service. Starting with .NET Framework 4, Microsoft implements wsHttpBinding instead of the BasicHttpBinding for providing security features on Web Services. BasicHttpBinding is very limited in its security settings. It provides only few security options in Message security such as user name tokens, Kerberos and X509 tokens. On the other hand, wsHttpBinding supports, by default, encryption for data transported using the Basic256 algorithmic suite. Furthermore, wsHttpBinding provides more in addition to the simple security features of BasicHttpBinding by supporting WS-* standards, and offering much more features such as security, reliable messaging, transaction support, duplex communications and many more. Some of the most important features that wsHttpBinding supports is WS-Security Policy, WS-Trust and WS-Secure Conversation.

Some of these security features can be easily noticed inside the part of the WSDL presented in the following figure:

```
            </sp:Wss11>
          - <sp:Trust10>
            - <wsp:Policy>
                <sp:MustSupportIssuedTokens/>
                <sp:RequireClientEntropy/>
                <sp:RequireServerEntropy/>
              </wsp:Policy>
            </sp:Trust10>
          </wsp:Policy>
        </sp:BootstrapPolicy>
      </wsp:Policy>
    </sp:SecureConversationToken>
  </wsp:Policy>
</sp:ProtectionToken>
- <sp:AlgorithmSuite>
  - <wsp:Policy>
      <sp:Basic256/>
    </wsp:Policy>
</sp:AlgorithmSuite>
- <sp:Layout>
  - <wsp:Policy>
      <sp:Strict/>
    </wsp:Policy>
</sp:Layout>
<sp:IncludeTimestamp/>
<sp:OnlySignEntireHeadersAndBody/>
</wsp:Policy>
</sp:SymmetricBinding>
```

Figure 47: A part of the WSDL of the protected SOAP service

Next table illustrates the differences between the BasicHttpBinding and wsHttpBinding.

Criteria	BasicHttpBinding	WsHttpBinding
Security support	This supports the old ASMX style, i.e. WS-BasicProfile 1.1.	This exposes web services using WS-* specifications.
Compatibility	This is aimed for clients who do not have .NET 3.0 installed and it supports wider ranges of clients. Many of the clients like Windows 2000 still do not run .NET 3.0. So older version of .NET can consume this service.	As its built using WS-* specifications, it does not support wider ranges of client and it cannot be consumed by older .NET version less than 3 version.
Soap version	SOAP 1.1	SOAP 1.2 and WS-Addressing specification.
Reliable messaging	Not supported. In other words, if a client fires two or three calls you really do not know if they will return back in the same order.	Supported as it supports WS-* specifications.
Default security options	By default, there is no security provided for messages when the client calls happen. In other words, data is sent as plain text.	As WsHttBinding supports WS-*, it has WS-Security enabled by default. So the data is not sent in plain text.
Security options	•None •Windows – default authentication •Basic •Certificate	•None •Transport •Message •Transport with message credentials

Table 5: BasicHttpBinding vs WsHttpBinding

Microsoft accompanies wsHttpBinding with mexHttpBinding for data transfer and metadata exchange with multiple endpoints. MexHttpBinding specifies the settings for a binding used for the WS-MetadataExchange (WS-MEX) message exchange over HTTP. MexBinding is essentially a WSHttpBinding binding with security disabled which supports most metadata requests exposing metadata endpoints. In order clients to retrieve metadata, they must be configured with a matching binding. I prefer to use Svcutil.exe to fetch the metadata and generate client code and configuration at design time. MexHttpBinding is not essential unless there are policies implemented which may change over time.

Message Layer Security

Authentication - Authorization

According to Microsoft, the use of Security token is mandatory when wsHttpBinding is used. Some of the other token types defined in the WS - Security Policy specification are the following: Username Tokens, X509 Tokens, Issued Tokens, Secure Conversation Tokens, SAML Tokens, Https Tokens. Security tokens are needed for the client in order to invoke the protected SOAP service. If a soap client wishes to invoke a protected SOAP web service, then the client must provide an authentication token in the scheme that the server is willing to accept. Generally, the IIS server does not accept Basic Auth tokens but instead favors NTLM and/or Kerberos/SPNEGO tokens.

Integrity

XML Digital Signatures

XML Digital Signatures is a standard that allows for specifying the syntax and processing rules for attaching digital signatures to XML documents. An XML digital signature takes data objects, calculates a *digest* (fixed-length representation of a variable-length stream), and places the result into the signature element.
In Windows Communication Framework security characteristics can be set in the public interface (ILaptops.cs). There, it is defined the protection level for each operation contract it is used among the following options: None, Sign or EncryptAndSign. In my Wcf case study, there are four Operation Contracts and each of them has Protection Level – Sign.

```
Example:    [OperationContract(ProtectionLevel = ProtectionLevel.Sign)]
            ModelObject[] GetModelsByPrice(int Mode);
```

Moving to the Message Contracts I set, again, signature as protection level for the message Header:

```
[MessageHeader(ProtectionLevel= ProtectionLevel.Sign)]
public int ID
{
    get { return _id; }
    set { _id = value; }
}

private string _model;
```

The message parts signed in the message body are presented as follows:

```
<sp:SignedParts>
  <sp:Body/>
  <sp:Header Name="To" Namespace="http://www.w3.org/2005/08/addressing"/>
  <sp:Header Name="From"   Namespace="http://www.w3.org/2005/08/addressing"/>
  <sp:Header Name="FaultTo" Namespace="http://www.w3.org/2005/08/addressing"/>
  <sp:Header Name="ReplyTo" Namespace="http://www.w3.org/2005/08/addressing"/>
  <sp:Header Name="MessageID" Namespace="http://www.w3.org/2005/08/addressing"/>
  <sp:Header Name="RelatesTo" Namespace="http://www.w3.org/2005/08/addressing"/>
  <sp:Header Name="Action" Namespace="http://www.w3.org/2005/08/addressing"/>
  </sp:SignedParts>
```

Next figure illustrates the SOAP message response structure for the Operation Contract `GetModelsByPrice` where it returns models according to one of the three price ranges that have been set. As we can notice, the whole SOAP message is digitally signed (the envelope). This is appeared in the message response where the letter “s” is in front of every tag. In addition to that, the SOAP Header is also signed having the `MustUnderstand` attribute.

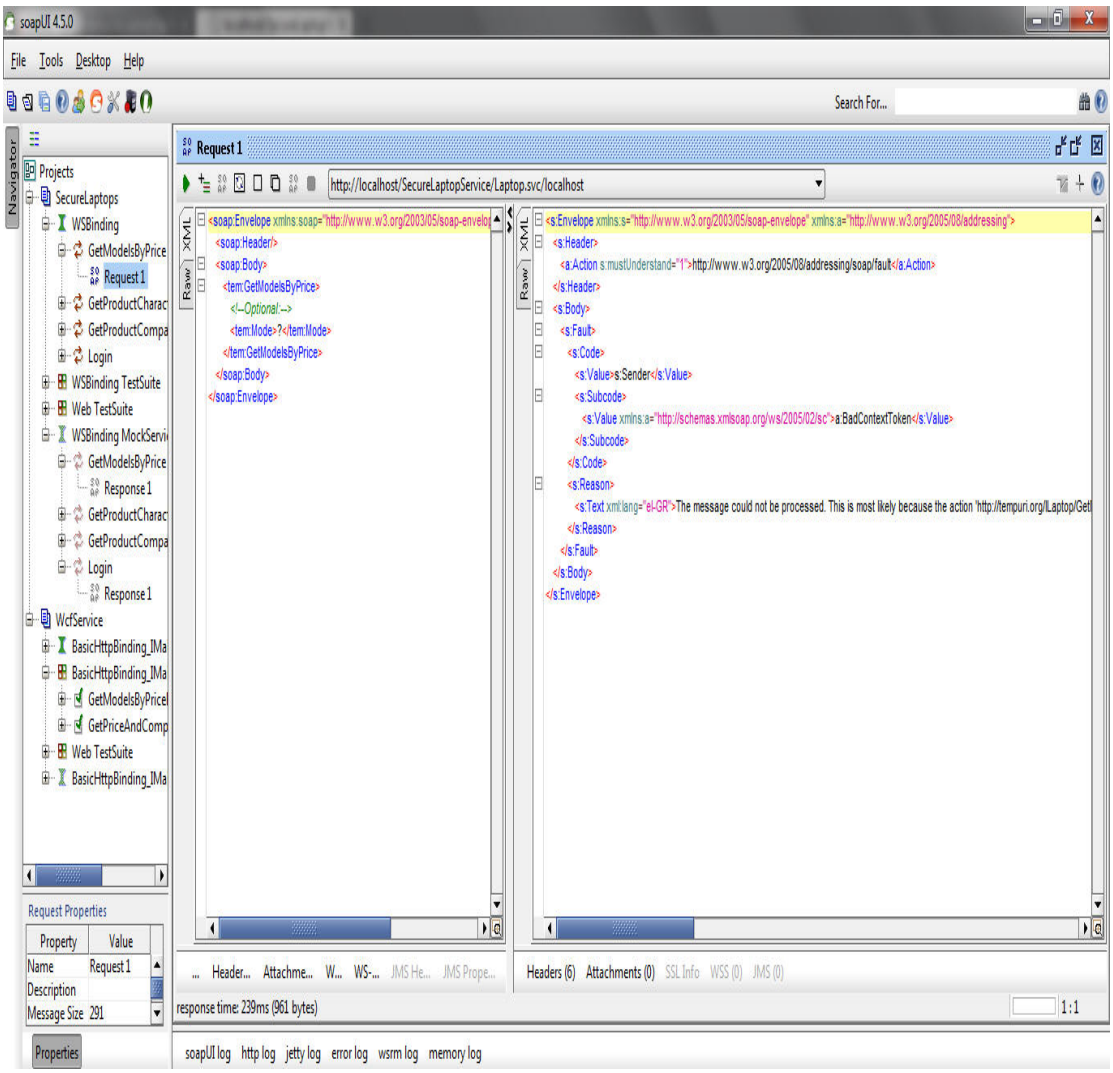


Figure 48: A secure SOAP message response structure

Encryption

Using similar logic with the digital signatures, XML encryption protects confidentiality by encrypting the whole or just parts of the SOAP message. In our example we can easily change Protection Level from sign to `EncryptAndSign` or just some parts of the SOAP message such as the `Body`. It is proposed to use XML encryption for the SOAP messages or parts of them that contain very sensitive information. Encryption is the best companion of signatures and therefore it is proposed to be used in conjunction with signatures as a more complete message security protection overall.

Input validation

The Wcf Service is not designed to allow data input. Instead, it uses drop-down menu for every operation where users make their choice. Furthermore, the service expects from the client to authenticate itself through the security token and also provide the same security through the `wsHttpBinding`.

As result, any other action prior to proper authentication is not allowed, appearing the following error:

"The message could not be processed. This is most likely because the action 'http://tempuri.org/ILaptop/GetModelsByPrice' is incorrect or because the message contains an invalid or expired security context token or because there is a mismatch between bindings. The security context token would be invalid if the service aborted the channel due to inactivity. To prevent the service from aborting idle sessions prematurely increase the Receive timeout on the service endpoint's binding.

Application Layer

IIS authentication and Access Control

Host-based authentication could be provided such as Integrated Windows Authentication disabling anonymous access. Integrated Windows authentication is easy to setup and does not require any development effort. It is all done in the IIS MMC and the web.config file. The current authentication type is NTLM which guarantees the privacy of the password since it is transmitted over the network the hash value of user's password. However, considering the SSL provided in transport layer and login – database authentication mechanism provided in the message layer, more authentication controls are not necessary.

Error Handling, Auditing and Logging

Error messages might leak information that leads to further attacks. Therefore, error messages should be disabled in ISS Web Server as well as any debug errors that might leak privacy related information. Instead, error messages could be replaced with a general custom message error, usually in html format. In addition, we should be able to track events and keep audit trails internally so that can be of assistance in case of abnormal or suspicious events. Finally, enabling IIS logging is always a good option for providing better logging traces and possibly preventing some types of attacks.

Last but not least, the use of an XML firewall would seem a perfect solution for first-line parameter defense. An XML firewall could be used to perform various security services at the message level such as auditing and Schema validation, authentication and authorization.

5. Conclusions

Web services are essential requirements for almost every activity today. Their importance, analyzed in detail in PART I, is huge. Their adoption and development will continue for many years. What has to be carefully considered is the security of Web Services. Web Service Security has to be further addressed and more specifically, for both developers and testers.

Developers on one hand must address security specifications on the SDLC during the Software Development Life Cycle (SDLC). Secure by design is a fundamental security aspect that every software developer should never neglect. They must secure from the code the Web Service or at least provide the necessary mechanisms in order for security to be addressed in the future according the needs of the service provider. This extremely important for Message Layer Security where security specifications such as XML encryption, XML signatures or data validation, can be addressed only during the code writing. Web Service Security should be addressed by developers as a unique and independent element no matter the hardware and software security of the host. The notion that developers write the code and then system administrators apply the security settings is totally wrong and disastrous. Furthermore, security should be addressed not only technically but also in terms of management and operational potential vulnerabilities. This includes, except for technology, people's awareness and existence of processes using adequate policies and standards for enabling security. In every business context, it is vital to understand the business requirements in order to rightfully access the business environment and apply the appropriate solutions.

On the other hand, penetration tester or generally any tester of Web Service Security (including the developers) should be provided with more robust and updated Web Service Testing Methodologies and tools. OWASP is moving to that direction by updating in the near future the "Web Service Testing Guide version 3" to version 4. However, current Web Service testing tools such as SoapUI, SOA Cleaner, Altova XML Spy and WCF Storm are still complicated, commercial-use oriented providing limited functionality for their "free - community editions". Furthermore, considering those testing solutions, they provide very less security focus and little automation. Tester's time is spend more in configuring tools and getting them running than in actual hacking.

Penetration testing is a common and useful technique for detecting known vulnerabilities. However, because it comes too late in the SDLC, it should never be considered as the only testing technique. According to OWASP, manual inspections and reviews, threat modeling and code review should pre-exist to penetration testing as a more holistic approach to security. Just as organizational policies, risk assessments, business continuity planning, and disaster recovery have become integral components in keeping organizations safe and secure, penetration testing needs to be included in this overall security plan as well. This presupposes existence of strong information security policy and readjustment when and where it is needed and also providing periodically but also unexpectedly information security audits and testing of web service implementations as well systems / applications and the network where they reside on. Systems should be always updated with the latest security patches and keep up-to- date for changes in technologies / specifications. Good configuration management is also very important that could save organizations a lot of trouble.

In conclusion, it is always important for everyone involved in security to remember that ”*Security is a process, not a product*”. Appropriate and organizational-specific security countermeasures should be applied according to the business environment, the number of security risks and their severity that affects business context as well as the cost and other implications that those risks may have on business continuity. Web Services exist long time and will continue to exist for an important period of time. We should, finally, consider Web Service Security as an eminent element and finally pay the appropriate attention.

Bibliography

[1] Dieter Fensel, Federico Michele Facca, Elena Simperl, Ioan Toma, 2011, "*Semantic Web Services*", Springer, e-ISBN 978-3-642-19193-0

[2] Stephen Potts, Mike Kopack, 2003, "*Sams Teach Youself Web Services in 24 hours*", Sams Publishing, ISBN 0-672-32515-2

[3] Steve Graham, Simeon Simeonov, Toufic Boubez, Doug Davis, Glen Daniels, Yuichi Nakamura, Ryo NeyamaSams, 2001, "*Building Web Services with Java™: Making Sense of XML, SOAP, WSDL, and UDDI*", Sams Publishing, ISBN 0-672-32181-5

[4] Robert Richards, 2006, "*Pro PHP XML and Web Services*", Apress, e-ISBN 978-1-59059-633-3

[5] Luciano Baresi, Elisabetta Di Nitto, 2007, "*Test and Analysis of Web Services*", Springer, e-ISBN 978-3-540-72911-2

[6] Microsoft MSDN training, 2002, "*Developing XML Web Services Using Microsoft® ASP.NET*". Microsoft Corporation, Course Number: 2524B, Part Number: X08-85030

[7] Ethan Cerami, 2002, "*Web Services Essentials - Distributed Applications with XML-RPC, SOAP, UDDI & WSDL*", O'Reilly, ISBN 0-596-00224-6

[8] David Chappell, Tyler Jewell, 2002, "*Java Web Services*", O'Reilly, ISBN 0-596-00269-6

[9] Doug Tidwell, James Snell, Pavel Kulchenko, 2001, "*Programming Web Services with SOAP*", O'Reilly, ISBN 0-596-00095-2

[10] Robert Englander, 2002, "*Java and SOAP*", O'Reilly, ISBN 0-596-00175-4

[11] Michael C. Daconta, Leo J. Obrst, Kevin T. Smith, 2003, "*The Semantic Web: A Guide to the Future of XML, Web Services, and Knowledge Management*", John Wiley & Sons, ISBN 0-471-43257-1

[12] Eric Newcomer, Greg Lomow, 2004, "*Understanding SOA with Web Services*", Addison Wesley Professional, ISBN 0-321-18086-0

[13] James McGovern, Sameer Tyagi, Michael Stevens, Sunil Matthew, 2003, "*Java Web Services Architecture*", Morgan Kaufmann Publishers, ISBN 1-558-60900-8

[14] Eric A. Marks, Mark J. Werrell, 2003, "*Executive's Guide to Web Services*", John Wiley & Sons, ISBN 0-471-26652-3

[15] Sanjiva Weerawarana, Francisco Curbera, Frank Leymann, Tony Storey, Donald F. Ferguson, 2005, "*Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*", Prentice Hall PTR,
ISBN 0-13-148874-0

[16] Thomas Mattern, Dan Woods, 2006, "*Enterprise SOA: Designing IT for Business Innovation*", O'Reilly, e-ISBN 978-0-59-610238-8

[17] Inderjeet Singh, Sean Brydon, Greg Murray, Vijay Ramachandran, Thierry Violleau, Beth Stearns, 2004, "*Designing Web Services with the J2EE™ 1.4 Platform JAX-RPC, SOAP, and XML Technologies*", Addison Wesley, ISBN 0-321-20521-9

[18] Richard Monson-Haefel, 2003, "*J2EE™ Web Services*", Addison Wesley, ISBN 0-321-14618-2

[19] Microsoft Press, 2003, "*Developing XML Web Services and Server Components with Microsoft Basic .NET and Microsoft Visual C# .NET*", Microsoft Corporation, ISBN 0-7356-1586-1

[20] Eric Newcomer, 2002, "*Understanding Web Services- XML, WSDL, SOAP and UDDI*", Independent Technology Guides, e-ISBN 978-0201750812
[21] Aphrodite Tsalgatidou, Thomi Pilioura, 2002, "*An Overview of Standards and Related Technology in Web Services*", Kluwer Academic Publishers, Distributed and Parallel Databases, vol. 12, pp. 135–162
[22] Microsoft MSDN, 2002, "*Security in a Web Services World: A Proposed Architecture and Roadmap*", available at: http://msdn.microsoft.com/en-us/library/ms977312.aspx [2012, June 10]
[23] David Pollmann, 2005, "*Programming INDIGO*", Microsoft Press, ISBN 0-7356-2151-9
[24] Paco Hope, Ben Walther, 2008, "Web Security Testing Cookbook, 1st Edition", O'Reilly Media, Inc., e-ISBN 978-0-596-51483-9
[25] William Stallings, 2003, "*Network Security Essentials – Applications and Standards Second Edition*", Prentice Hall, ISBN 0-13- 035128-8
[26] Russell Lusignan, Oliver Steudler, Jacques Allison, 2000, "*Managing Cisco Network Security: Building Rock-Solid Networks*", Syngress Publishing, ISBN 1-928994-17-2
[27] Douglas K. Barry, 2003, "Web Services and Service-Oriented Architectures - The Savvy Manager's Guide", Elsevier, e-ISBN 978-1-55860-906-8
[28] J.D. Meier, Alex Mackman, Srinath Vasireddy, Michael Dunner, Ray Escamilla, Anandha Murukan, 2003, "*Improving Web Application Security - Threats and Countermeasures*", Microsoft Corporation, Patterns and Practices, For current list of titles visit: msdn.microsoft.com/practices
[29] Esmiralda Moradian , Anne Håkansson, 2006, "Possible attacks on XML Web Services", IJCSNS International Journal of Computer Science and 154 Network Security, VOL.6 No.1B
[30] OWASP, 2005, "*A Guide to Building Secure Web Applications and Web Services*", The Open Web Application Security Project (OWASP), 2.0 Black Hat Edition, available at : www.owasp.org [2012, June 10]
[31] OASIS, 2006, "*Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)*", OASIS Standard incorporating Approved Errata, available at: http://docs.oasis-open.org/wss/v1.1/ [2012, June 10]
[32] Frederick Hirsch, John Kemp, Jani Ilkka, 2006, "*Mobile Web Services - Architecture and Implementation*", John Wiley & Sons, ISBN 047-001-596-9
[33] WS-I, 2005, *"Security Challenges, Threats and Countermeasures Version 1.0 - Final Material",* available at: http://www.ws-i.org/Profiles/BasicSecurity/SecurityChallenges-1.0-20050507.doc [2012, June 10]
[34] Judith Hurwitz, Robin Bloor, Marcia Kaufman, Dr. Fern Halper, 2009, "*Service Oriented Architecture For Dummies®, 2nd IBM Limited Edition*", Wiley Publishing, e-ISBN 978-0-470-52549-4
[35] Heather Kreger, Vince Brunssen, 2012, "*Service-oriented architecture (SOA) standards - IBM standards*", IBM Corporation, available at*: http://www.ibm.com/developerworks/*
[36] OWASP project, 2004, "*The Ten Most Critical Web Application Security Vulnerabilities*", available at: www.owasp.org, [2012, June 10]
[37] OWASP project, 2008, "*OWASP Testing Guide v.3*", available at: www.owasp.org, [2012, June 10]

[38] OWASP project, 2010, "*OWASP Top 10 – 2010: The Ten Most Critical Web Application Security Risks*", available at: www.owasp.org [2012, June 10]
[39] Sacha Faust, 2005, "*Web Application Testing with SPI Fuzzer*", SPI Dynamics Inc.
[40] IATAC, 2011, "*Vulnerability Assessment*", Sixth Edition, Information Assurance Tools Report
[41] Nancy Muir, Ian Kimbell, 2010, "*Discover SAP*", Galileo Press, e-ISBN 978-1-59229-320-9
[42] W3C, 2004, "*Web Services Architecture*", W3C Working Group Note 11 February 2004, available at: http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/ [2012, June 10]
[43] E. Kleiner, A.W. Roscoe, 2006, "*On the Relationship Between Web Services Security and Traditional Protocols*", Elsevier, Electronic Notes in Theoretical Computer Science, vol. 155, pp. 583–603
[44] Walid Negm, 2004, "*Anatomy o f a Web Services Attack – A guide to threats and preventative countermeasures*", Forum Systems Inc.
[45] Don Patterson, InfoSec Reading Room, 2007, "*XML Firewall Architecture and Best Practices for Configuration and Auditing*", SANS Institute
[46] Kevin Spett, 2002, "*SQL Injection Are Your Web Applications Vulnerable?*", SPI Dynamics
[47] Kevin Beaver, Peter T.Davis, 2005, "*Hacking Wireless Networks For Dummies*", Wiley Publishing, e-ISBN 978-0-7645-9730-5
[48] Alex Stamos, Scott Stender, 2005, "*Attacking Web Services – The next generation of Vulnerable Enterprise Applications*", Information Security Partners, available at: www.isecpartners.com [2012, June 10]
[49] Karen Scarfone Murugiah Souppaya, Amanda Cody, Angela Orebaugh, 2008, "*Technical Guide to Information Security Testing and Assessment*", National Institute of Standards and Technology, Special Publication 800-115
[50] Chan Tuck Wai, InfoSec Reading Room, 2002, "*Conducting a Penetration Test on an Organization*", SANS Institute
[51] Patrick Engebretson, 2011, "*The Basics of Hacking and Penetration Testing - Ethical Hacking and Penetration Testing Made Easy*", Elsevier, ISBN 978-1-59749-655-1
[52] Johnny Long, 2005, "*Google Hacking for Penetration Testers*", Syngress Publishing, ISBN 1-931836-36-1
[53] Andrew Whitaker, Daniel P. Newman, 2006, "*Penetration Testing and Network Defense*", Cisco Press, ISBN 1-58705-208-3
[54] T. J. Klevinsky, Scott Laliberte, Ajay Gupta, 2002, "*Hack I.T.: Security Through Penetration Testing*", Addison Wesley, ISBN 0-201-71956-8
[55] Thomas Wilhelm, 2010, "*Professional Penetration Testing: Creating and Operating a Formal Hacking Lab*", Elsevier, e-ISBN 978-1-59749-425-0
[56] Enumeration of all known web service attacks available at: www.ws-atacks.org