

IAPF: A Framework for Enhancing Web Services Security

Navya Sidharth and Jigang Liu
Metropolitan State University
700 East Seventh Street
Saint Paul, Minnesota 55106, USA
sidharna@go.metrostate.edu , jigang.liu@metrostate.edu

ABSTRACT

The applicability of the security protocols, such as WS-Security, WS-Trust, WS-SecureConversation, WS-Federation, WS-Authorization, and WS-SecurityPolicy, is limited as they only protect SOA (Service Oriented Architecture) communication between two trusted parties with an established security association. The pervasiveness of web services and SOAP API that can be invoked by anonymous consumers introduces security vulnerabilities are not addressed by the existing standards. In this paper, an Integrated Application and Protocol-based Framework is proposed to tackle the existing WS security problems. The proposed IAPF techniques are envisioned to be a part of the design and implementation structure of a web service endpoint within the application and transaction handling logic of the SOAP/web service producer. These techniques will empower application level web services developers to design and implement SOA producers to the IAPF standard to firstly prevent DoS and DDoS based attacks and secondly mitigate the effects of these attacks

Keywords

Web services, WS-Security, UDDI, WSDL, DoS and SOAP

1. INTRODUCTION

According to the W3C, a **Web service and SOA system** primarily supports interoperable machine-to-machine interaction over heterogeneous enterprise networks. The SOA and web services technologies were standardized by the World Wide Web consortium to address weaknesses in traditional distributed messaging techniques such as COM, RMI, DCOM and CORBA. Traditional distributed messaging does not provide the capability to connect heterogeneous systems separated by firewalls across organizational and enterprise boundaries seamlessly. This is because firewalls normally permit only HTTP and SMTP traffic while blocking traffic over other protocols to protect against DoS attacks. This has prevented enterprises from integrating systems that reside over disparate LANs. The SOA framework allows for the exposing of services instead of plain objects. The communication between these

services are achieved using SOAP or other XML messaging formats over protocols such as HTTP instead of using protocols such as RMI.

The various components or technologies of a web service include: UDDI (Universal Discovery Description and Integration), WSDL (Web Services Description Language) and SOAP (Simple Object Access Protocol). UDDI is a specification of a Web service registry that is used extensively in the industry. UDDI registries can be hosted internally within an enterprise or a group of enterprises and used for looking up Web services, much the same way as a LDAP directory. A UDDI registry is itself a Web service. Therefore searching and publishing to the registry can be done using SOAP messages. The interface provided by a Web service is often defined using an XML format called WSDL. A WSDL description can be shared with a Web service client in many ways, but the recommended way is to use a registry to publish it. The message format used in a SOA based system is usually SOAP (Simple Object Access Protocol). SOA also allows for services implemented on a platform such as J2EE to communicate effortlessly with a service implemented in .NET or Python. Protocols such as a RMI and CORBA are specific to certain languages and compilers and do not permit the construction of distributed systems over heterogeneous platforms.

The seamless interoperability characteristics of web services also introduce security concerns that do not exist in traditional distributed messaging techniques like RMI and CORBA. This is because the SOAP based XML messages are not firewall proof and this could lead to intruders gaining access to sensitive systems using the interfaces provided by the WSDL files for service access. Moreover, traditional websites designed using non-SOA frameworks allow users to interact with an application remotely through a web browser but a web service allows other applications to interact directly with it. This makes a web services more vulnerable to security problems.

Security threats such as XML and SQL injections are specific to web services and can lead to severe DoS of the affected systems. The ease of use, platform independence,

use of HTTP and other attributes that make Web services attractive also make them great targets for attack. Protecting web services requires developers, system administrators, and management to work together. Web application security must be a continually evolving process that evolves to counter the ever changing methodologies used by hackers. The next section provides additional details about WSDL, UDDI and SOAP and outlines the security vulnerabilities that could potentially occur due to the design principles followed in formulating these technologies.

2. Overview of Web Services Technologies

The following paragraphs provide an overview of the various technologies [1] used within a web services based system.

Universal Discovery Description and Integration (UDDI): UDDI is a specification for defining a registry of information for discovery of web services. This specification defines a means to publish and discover information about web services, including WSDL files. It creates a standard interoperable platform so that it can enable companies and applications to find and use Web services dynamically over the Internet. After browsing through an UDDI registry for information about available web services, the interfaces for the selected service can be viewed in a Web Services Description Language (WSDL) file, and an appropriate SOAP message can be sent to the service.

Web Service Description Language (WSDL): Web Services Description Language (WSDL) is an XML format for describing web services. WSDL describes the structure of a specific service using XML-formatted data. The information provided includes the following:

- Service name and location
- Methods name and argument type
- Return value and type
- Documentation about the service

Through a WSDL description, a client application can determine the location of the remote web service, the functions it implements, as well as how to access and use each function. WSDL files are typically stored in registries that can be searched by potential clients to locate web service implementations of desired capabilities.

Simple Object Access Protocol (SOAP): SOAP is a lightweight and simple XML-based protocol designed to exchange structured and type information on the Web. This information exchange mechanism can be used to send messages between applications and to implement remote procedure calls. SOAP is a platform and application language independent mechanism that is used for

expressing application semantics that can be understood by applications irrespective of their implementation. SOAP is usually used in conjunction with HTTP. In the SOAP XML schema, the envelope element is always the root element of a SOAP message. Furthermore, there are two types of envelopes, Request and Response [4].

The Request envelope contains the information required to process the remote call. Each request message contains a message header and a message body. The header stores processing information such as message routing, requirements information, and security information. The body of the message stores the information required to process the call like the name of the service called, location of the service and the parameter names and data passed to the service.

The response envelope is used to return data regarding a previous request message sent to the server. The information in a successful request contains the method called, the return value type, and the return data. This response envelope is used for responses generated under normal conditions. If an error occurs on the server side, a fault response message is used to return data to the client. When an error occurs on the server side, the SOAP service returns an envelope type of Fault, which may contain more specific information about the error.

3. Web Services Attacks

Attacks on web services can be broadly classified into attacks that use vulnerabilities in WSDL to get information about various methods and parameters needed for the attack. UDDI can also be used to get information on various web services and their access points. The SOAP messages can also be attacked.

Attacks Leveraging UDDI Vulnerabilities: Web service consumers query the Universal Business Registry (UBR) to determine the details of the services to be used. Using UBR, a consumer can find out what the service is and where it resides. By analyzing the UDDI, a consumer can discover information about the service name, services provided by the web service publisher and the details of the WSDL file used to invoke the service as well as the location of the service. In fact, the UDDI protocol is very similar to a “whois” server [2]. Queries sent to a UDDI server result in responses being sent back with the required information. Web services are usually registered on a UDDI server using one of the following structures:

1. Business Entity
2. Business Service
3. Binding Template
4. Technical Model (tModel)

The Universal Business Registry's (UBRs) on the UDDI servers are created by various companies such as Microsoft, IBM, SAP etc. These companies replicate their data with one another. There is a set of APIs that help in querying this registry. Attackers can use each of these APIs to query the UBR for specific information. For example, if an attacker is looking for information on a certain company, it can footprint the company using all these APIs and see what information can be extracted. These APIs usually work on HTTP with SOAP. The WSDL specification is already published and can be used by these inquire APIs to extract the requisite information. Different sets of toolkits and SDKs are published and developed by various companies such as Microsoft, Sun etc. These toolkits can be used to write programs to enumerate or just invoke the APIs using SOAP on simple TCP/IP clients.

Attacks using Weaknesses in WSDL: The handlers for the SOAP messages can be enumerated by profiling the WSDL file and determining the methods and the input and output parameters of each method [3]. Once an attacker has that information, it can make requests to generate SOAP faults and other faults to determine the type of database exception being thrown, etc. In short this attack can be used to phish for information from the backend databases. The WSDL stores all critical information about web services and must be shared with rest of the world on the Internet. This is the only way others can invoke web services over a network. Utilities are available in Java as well as .NET to convert the request and response interfaces into a WSDL file and vice versa. Therefore, by carefully analyzing the interfaces within a WSDL file, guesses can be made about the underlying implementation of the classes and interfaces that are used for handling the SOAP messages.

For instance, an interface named setLatestStockData could be defined in a WSDL file as follows:

```
<element name="setLatestStockData">
  <complexType>
    <sequence>
      <element name="price" type="xsd:double"/>
      <element name="volume" type="xsd:long"/>
    </sequence>
  </complexType>
</element>
```

By looking at this piece of code from a WSDL file, we know that the underlying method named setLatestStockData needs two parameters, price of type double and volume of type long. In the same way, an attacker can look at all the methods in the WSDL file, and make a chart of all the methods with their inputs and outputs. This can be used to make SOAP requests to

generate faults to get more information about the web service.

Attacks on the SOAP Messages: Attacks on the SOAP protocol include parameter tampering and replay attacks.

Parameter Tampering: In parameter tampering or SQL injection [4,5], the web service application may not perform any input validation. This could lead to input being injected into the SQL to create problems with returns from the database queries. Middleware implementations in J2EE and .NET that do not rigorously examine input strings are extremely susceptible to parameter tampering. For example, if an attacker adds a single quotation mark to the input string and the application accepts the input and passes the user-supplied data un-sanitized to an SQL statement, then the application is susceptible to SQL injection. The first step in a parameter tampering attack is to determine if the web service application is performing any type of input validation. To generate a good set of inputs to use for testing, the WSDL document must be rigorously analyzed. In addition to specifying the data type of each parameter, the WSDL document will often contain comments about the purpose of those parameters. It can also be deduced if the service gets its information from a data repository (a database or LDAP server), so the service might well be vulnerable to SQL or LDAP injection. By carefully examining the SOAP/other fault string returned by the server, we can determine the type of database server being used in the backend. A closer look at the fault string will also reveal more information about some of the internal processing the web service performs and where the problem occurs. All this information can be used to strategically formulate targeted attacks towards the vulnerable systems.

Replay Attack: This attack is similar to the "network ping of death" in which a hacker can issue repetitive SOAP message requests in a bid to overload a Web service. This type of network activity will not be detected as an intrusion because the source IP is valid, the network packet behavior is valid and the HTTP request is well formed. However, the business behavior is not legitimate and constitutes an XML-based intrusion. In this manner, a completely valid XML payload can be used to issue a denial of service attack.

WsChess – An Open Source Tool for Attacking Vulnerabilities in Web Services: A few open source tools have been implemented for probing vulnerabilities in web services. WsChess [6,7] is a tool that is used for assessing web services security. Using wsChess, the following steps can be used to formulate attacks on web services.

1. A Universal Business Registry (UBR) can be queried using wsPawn, a component of wsChess. wsPawn can be specifically used to query the UDDI for services by business name.

In addition to the querying capabilities, WsPawn also provides services to find registered web services and their access points and retrieve information from the public UDDI.

2. The wsPawn tool is used to query the UDDI on a service key. A set of matching WSDLs is returned by this query.

Another way of finding the WSDL file is to search for the specific WSDL on a search engine like google. For example if we search for the string “amazon webservicess wsdl” on google, a URL pointing to the WSDL is returned.

3. Once we have the location of the WSDL file, we can examine WSDL for vulnerabilities. The wsKnight tool helps in profiling the web service from its corresponding WSDL. It also allows you to invoke methods and intercept them before they go on the wire to the target, so that you can manipulate the SOAP envelope if needed. Once the methods and their corresponding inputs and outputs are profiled, the malicious user can go on to sending SOAP messages to attack the web services.

Other Attacks: Coercive parsing, recursive payloads, oversize payloads, schema poisoning and routing detours [8] are some additional forms of attacks that could be launched against a web services based system.

Recursive Payloads: Elements are usually nested within an XML document. Malicious attackers can easily create documents that attempts to stress and break an XML parser by creating a document that is 10,000 or 100,000 elements deep. This could lead to common DOM and SAX XML parsers to recursively parse the elements and cause memory related issues such as buffer overflows in web services based systems.

Oversize Payloads: XML files usually inflate verbose binary data which are usually the norm in many enterprises. Converting batch processes to an SOA based real time transfer process can easily lead to XML documents that are hundreds of megabytes in size. These oversize payloads could cause DOM based parsers to malfunction. DOM based parsers have to load the entire document into memory and this might lead to memory overflow issues.

Schema Poisoning: XML Schemas provide formatting instructions for parsers when interpreting XML documents.

Schemas are used for all of the major XML standard grammars coming out of OASIS. As most webservicess implementations validate XML instances against a schema before processing, an attacker might replace an existing schema with a malicious one to cause the webservice to fail. The attacker might change the data type to cause severe denial of service attacks by using schema poisoning techniques.

Routing Detours: The WS-Routing specification provides a way to direct XML traffic through a complex environment. It operates by allowing an interim way station in an XML path to assign routing instructions to an XML document. Companies such as Reactivity make these XML gateways that can perform encryption and decryption of WS-Security documents. A compromised or malicious gateway may participate in a man-in-the-middle attack by inserting bogus routing instructions to point a confidential document to a malicious location. From that location, it may be possible to forward on the document, after stripping out the malicious instructions, to its original destination. SOAP intermediaries that are extensively used in a web services environment could also participate in malicious activity by causing routing detours.

Coercive Parsing: As web services and SOAP based systems are primarily built using XML, they are susceptible to all kinds of generic XML attacks. By directly attacking the XML using well known XML based attacks, intruders could create attacks of a malicious nature.

4. Protecting Against Attacks

The Integrated Application and Protocol Level Framework (IAPF) techniques approach the attacks using a four pronged approach. The first three facets are primarily concerned with protecting against attacks on the UDDI, WSDL, and SOAP protocols. The fourth facet addresses techniques to protect against attacks that could be launched on openly available web services that are invoked by untrusted web service consumers.

a) Protecting against Attacks on UDDI: UDDI is the starting point for attacking web services. All the web services are registered on the UDDI registries [2]. The advantage of this is that web services can be searched for on these registries. By looking at the UDDI one can find information about the service name, services provided by the web service publisher and the details of the WSDL file used to invoke the service as well as the location of the service. Once the user has this information they can invoke the specific web service in their applications. This allows software and services from different companies and locations to be combined easily to provide an integrated service. The disadvantage of this feature is that a malicious

user or program can also access these registries to gain information about a web service [9]. For example, if a malicious user is looking for information on a certain company, they can footprint the company using all these UDDI and see what information can be extracted.

Some of the solutions to prevent information from UDDI registries to be accessed by malicious users are as follows: The UDDI registries need not be visible or accessible to every user. The registries can have registered users who have access to the registry; unauthorized users will not be able to access the registry. The UDDI registry can also introduce access control mechanisms on the users, so that only some of the registered users gain access to a particular UDDI like potential clients of a web service. Since the UDDI registry will not be visible to all users, the threat of malicious users' footprinting a service is greatly reduced but not altogether removed.

b) Protecting against Attacks on the WSDL Protocol: WSDL files have all the sensitive information needed to attack a web service. The solution to protect against these attacks is to limit the access of UDDI registries will limit the number of people who can look at the WSDL location by footprinting the web services. If a malicious user manages to gain access to the WSDL by other means other than querying the UDDI registry. The malicious user or program can look at the WSDL file of the web service. The WSDL file contains all the information necessary to use or attack a web service. The WSDL file can be used for enumerating and profiling a web service [3].

A solution to prevent unwarranted information from being present in the WSDL file is for the developers to manually check the WSDL files and make sure unnecessary information is not present that may aid a malicious user in attacking the web service.

For example WSDL files might expose the debugging methods written by the developer of the web service [9]. These methods were necessary for the developer to debug and get the web service to work as required, but might have information that can be used by a malicious user to attack the web service. Since the developer is in a better position to decipher whether certain information on the WSDL file is unnecessary for the users or clients, it is better that the developer manually goes through the WSDL file to remove any unnecessary information. If the sensitive information that can be used to attack a web service is not visible, a malicious user will not have the necessary information to attack the web service.

c) Protecting against Attacks on the SOAP Protocol

There are numerous attacks that can be launched using the vulnerabilities in the SOAP protocol. Several application

layer as well as protocol level features are used in the Integrated Application and Protocol Framework (IAPF) to protect web services from malicious attackers.

Protecting Web Services from Parameter Tampering

Attacks: Parameter tampering attacks can be launched by using either SQL injection or LDAP injection [4]. SOAP is used in conjunction with HTTP and thus is designed to allow SOAP based XML messages to pass through firewalls without being processed. This could lead to intruders gaining access to sensitive systems using the interfaces provided by the WSDL files for service access by sending malicious SOAP messages. There are several ways to prevent malicious SOAP messages from accessing the web service server or the application code. Microsoft's .Net framework provides two interfaces called IHttpHandler and IHttpModule [11]. Web applications can be built using these interfaces, which provide functionalities to check a HTTP request before it reaches the application code or server.

Leveraging In-built Features in Web Servers to Prevent

Attacks: The Apache web server also provides functionality to prevent malicious HTTP requests from accessing the web service server or the application code. This feature can be used by developers who are using Java to deploy the web services. The mod_security [12] feature of the Apache server makes use of regular expressions to defend the web service inputs. By using regular expressions we can match patterns of malicious requests or inputs for example, we can use regular expressions to filter out meta characters similar to % and like. This will help in preventing SQL Injection attacks against the web services. The mod_security feature of the Apache server can also be used to prevent unnecessary information from getting to the end user through the fault response messages. Another advantage of mod_security is that it can be used by developers and administrators without actually having to modify the source code. So this can be used by administrators to provide an extra layer of security to the web services.

Attack Prevention using Web Service Firewalls:

Another solution to attack prevention is to design a web service firewall [10] that protects the web service server from any malicious SOAP messages. By using this mechanism, the malicious SOAP messages can be detected even before they reach the web service server or application. Malicious SOAP messages are messages that do not adhere to the web service specifications. There can be a number of ways to check the validity of the SOAP messages. One way to check if the SOAP messages are valid is to check the XML based message against the specified XML schema. The web service firewall can be

used to perform several checks on the SOAP messages. The SOAP messages can be first checked to see if they conform to the security policy implemented by the web service server. This includes checking the security tokens in the SOAP message and also verifying the integrity and authentication of the SOAP messages.

Attack Prevention using WS-Security Policy: Yet another solution is to use WS-Security Policy [13]. In a typical SOA, where the client and the service may not be in the same security domain, policies enforce security rules on the outgoing and incoming messages. The goal of WS-Security Policy is to enable a web services participant to engage in secure exchange of messages with web service. WS-Security Policy is a Microsoft, IBM, Verisign, and RSA Security specification that describes how senders and receivers can specify their security requirements and capabilities. WS-Security Policy is based on WS-Policy and also defines a mechanism for attaching or associating service policies with SOAP messages. WS-Security Policy Language is a declarative XML-based language for security policies between web service clients and web service servers. A web service server can describe its security policy using WS-Security-Policy documents and provide them to its clients. WS-Security Policy is designed to work with the general Web Services framework including WSDL service descriptions, UDDI business Services and binding Templates and SOAP message structure and message processing model, and WS-Security Policy should be applicable to any version of SOAP. SOAP messages sent between web service clients and servers should include the WS-Security elements in accordance with the agreed upon security policy. In order to protect the web service servers SOAP messages need to be checked to make sure they are not a threat to the security of the web services. SOAP messages that do not adhere to the Web Service security specification that is laid down in the WS-Security-Policy document can be flagged as errors, this helps improve the web service server security and protects it from malicious SOAP messages.

d) Protecting Against Attacks on Openly Available Web Services

The techniques presented previously for making SOAP messages secure are only applicable when the communication is between two trusted partners. When the two partners trust each other they can use public key cryptography as well as symmetric key encryption within a web services security standard like WS-Security to obtain a truly secure solution. In fact, many commercial organizations use advanced hardware acceleration based security solutions from vendors like Reactivity (<http://www.reactivity.com>) to setup secure communication between their trusted partners. However, this technique cannot be used in a situation where a web services

consumer cannot be trusted. A real-world example is presented in the following paragraph to illustrate this case. For instance, Salesforce.com exposes a SOAP API that can be used by any web services consumer to invoke various services.

The following paragraph provides a sample SOAP request that can potentially be made to Salesforce.com's SOAP API.

```
<?xml version="1.0" encoding="utf-8"?>
  <soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope"
    />
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://
www.w3.org/2001/XMLSchema">
    <soap:Header>
      <SessionHeader
        xmlns="urn:enterprise.soap.sforce.com">

<sessionId>z1E_NI80RXGyM48GB2etyLuR8DBelyRk_MTjFe0
xPed0_jXINKOSU</sessionId>
      </SessionHeader>
    </soap:Header>
    <soap:Body>
      <getUserInfo xmlns="urn:enterprise.soap.sforce.com" />
    </soap:Body>
  </soap:Envelope>
```

This SOAP web service is also available on the following URL:

http://www.salesforce.com/us/developer/resources/soap/sforce60/sforce_API_messages_getUserInfo.html.

Creation of a webservices consumer for this “getuserinfo” webservice from Salesforce.com is trivial. Technologies such as XMLbeans can be used on .NET, J2EE or Python to rapidly create a web service consumer for this service. As a security association does not exist between “Salesforce.com and the web service consumer” a variety of attacks are possible. Protecting against these attacks is a challenge. This challenge could be analogous to the challenge of developing protection against Distributed Denial of Service (DDoS) attacks that could potentially be launched on lower level protocols like TCP/IP to cause service outage. A DoS or DDoS attack akin to the network ping of death or a TCP SYN flood attack where several requests are made for the same SOAP web service could potentially cause an outage of the web service.

Protection against these kinds of attacks must be implemented within the application and transaction handling logic of the SOAP/web service producer. For instance, if by mistake or negligence a database exception

is propagated to a SOAP based exception/fault, the attacker could determine database structure and vulnerabilities to launch a SQL injection attack within the web service request. Several other commercial organizations expose web services in a similar fashion. Google adworlds and Amazon.com are notable examples. Further research is needed in this arena to develop a framework so that application level web services developers can design and implement systems to standard that prevents DoS and DDoS based attacks on web service producers.

The following paragraph presents an example to illustrate how an attack can be launched against web services that are exposed for general consumption, and techniques that are present within the IAPF to protect against this attack.

Parameter Tampering - SQL Injection Attack:

For example, a method named setLatestStockData is represented as follows in a WSDL file.

```
<element name="setLatestStockData">
  <complexType>
    <sequence>
      <element name="name" type="xsd:string"/>
      <element name="price" type="xsd:double"/>
      <element name="volume" type="xsd:long"/>
    </sequence>
  </complexType>
</element>
```

To generate a good set of inputs to use for testing, the WSDL document must be rigorously analyzed. In addition to specifying the data type of each parameter, the WSDL document will often contain comments about the purpose of those parameters. For example in the WSDL document snippet provided above if we write a SOAP request message

```
<SOAP-ENV:Body>
  <SOAPSDK4:SetLatestStockPrice
xmlns:SOAPSDK4="http://localhost/SetStockPrice/">
  <SOAPSDK4:name>%maz%</SAOPSD4:name>
  <SOAPSDK4:price>120.25</SAOPSD4:price>
  <SOAPSDK4:volume>1</SAOPSD4:volume>
  </SOAPSDK4:SetLatestStockPrice>
</SOAP-ENV:Body>
```

Suppose at the backend we have a query that just extracts the information from this request. If we have a table called stock price that maintains the price information, then a query like “update stock_price set price = 120.25 , volume

= 1 where name =%ma%”. Then in this case all stocks that have a name similar to maz e.g amazon.com Inc., Amazon Biotech Inc. and Amazon oil and energy Inc. will have their price and volume set.

Techniques within the IAPF framework will examine the SOAP message for common SQL injection tactics and prevent the SOAP message from reaching the server and in turn executing this statement on the database.

5. The IAPF Approach - Details

The Integrated Application and Protocol Framework (IAPF) is organized in a sequence of steps to enable web services security implementers; as well as web services implementers to formulate an integrated approach to securing web services. Figure 1 provides a system architecture based view of the IAPF approach.

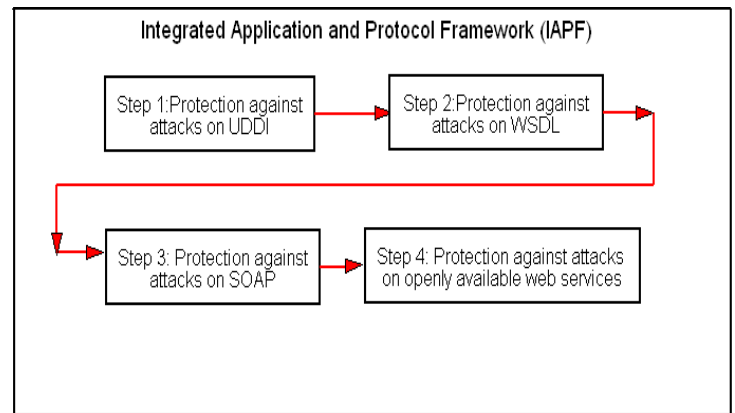


Figure 1: IAPF System Architecture

In the IAPF approach, the first step involves providing protection against vulnerabilities in the UDDI protocol. The second step involves protecting vulnerabilities in the WSDL protocol. In the third step, comprehensive protections are built using techniques such as WS-Security to protect vulnerabilities in the SOAP protocol for end to end communication between two entities. In the fourth step, protection mechanisms are built to protect SOAP web services that need to be exposed openly to third party consumers.

The IAPF techniques present a comprehensive solution that can be used by web services security implementers as well as web services application developers to provide secure and safe web service access.

6. Conclusion and Future Work

The ease of use, platform independence, use of HTTP and other attributes that make Web services attractive also make them great targets for attack. The various

components of web services like UDDI, WSDL and SOAP allow various web services to be integrated to provide interoperability between web services. The seamless interoperability characteristics of web services also make them susceptible to security threats that are unique to web services. Protecting web services requires developers, system administrators and the management to work together. Web services security must be a continually evolving process so as to counter the new attacks that can be discovered in the future. In this paper, an outline of some of the current attacks that can be launched to exploit vulnerabilities within web services has been presented along with a comprehensive Integrated Application and Protocol based Framework (IAPF) for preventing these attacks. Following the Guide to Secure Web Services [14] published by NIST recently, we plan to implement a prototype WS application to further assess and then improve the IAPF framework..

7. REFERENCES

- [1] Developing Enterprise Web Services, An Architect's guide- Sandeep Chatterjee and James Webber, Prentice Hall, ISBN-10: 0131401602, 2003
- [2] Web Services- Attacks and Defense, Information Gathering Methods: Footprints, Discovery and Fingerprints-Shreeraj Shah.: http://www.net-square.com/whitepapers/WebServices_Info_Gathering.pdf Site last accessed 09/24/06
- [3] Web Services: Enumeration and Profiling-Shreeraj Shah, http://www.net-square.com/whitepapers/WebServices_Profiling.pdf Site last accessed 09/24/06
- [4] SOAP Web Services Attacks: Are your web applications vulnerable? Sacha Faust, SPI Dynamics, 2005 http://www.spidynamics.com/whitepapers/SOAP_Web_Security.pdf Site last accessed 04/28/07
- [5] Blind SQL Injection: Are your web applications vulnerable? Sacha Faust, SPI Dynamics, 2005 <http://www.spidynamics.com/whitepapers/WhitepaperSQLInjection.pdf> Site last accessed 09/24/06
- [6] WsChess- Web Services Assessment and Defense Toolkit: Site <http://net-square.com/wsches/index.html> Site last accessed : 09/24/06
- [7] Introduction to Assessing and Securing Web Services- Christoff Breytenbach http://64.233.167.104/search?q=cache:AJ1Wb0X5zwMJ:icsa.cs.up.ac.za/issa/2005/Proceedings/Full/058_Article.pdf+Introduction+to+Assessing+and+Securing+Web+services+%22Christoff+Breytenbach%22&hl=en&ct=clnk&cd=6&gl=us Site Last accessed 09/24/06
- [8] Attacking and Defending Web Services, Pete Lindstrom, January 2004 http://forumsystems.com/papers/Attacking_and_Defending_WS.pdf Site last accessed 09/24/06
- [9] Attacking web services-The Next generation of Vulnerable Enterprise Apps, Alex Stamos, Scott Stender. BlackHat 2005 <http://www.blackhat.com/presentations/bh-usa-05/bh-us-05-stamos.pdf> Site last accessed 09/24/06
- [10] Event Based SOAP Message Validation for WS-Security-Policy-Enriched Web Services. Nils Gruscha, Norbert Luttenberger, and Ralph Herkenhoner. <http://www1.ucmss.com/books/LFS/CSREA2006/SWW4856.pdf> Site last accessed 09/24/06
- [11] Web application defense at the gates- Leveraging IHTTPModule – Shreeraj Shah. Site: http://net-square.com/whitepapers/WebApp_HTTPMod.pdf. Site last accessed: 09/24/06
- [12] Defending web Services using Mod Security(Apache)- Methodolgy and Filtering Techniques-Shreeraj shah Site: <http://net-square.com/whitepapers/Defending-web-services.pdf>. Site last accessed: 09/24/06
- [13] Web Services Security policy language-WS-Security-Policy Version 1.1, July 2005. Giovanni Della-Libera, and *et. al.*
- [14] Guide to Secure Web Services (Draft) , NIST, September 2006 <http://csrc.nist.gov/publications/drafts/Draft-SP800-95.pdf> Site last accessed 04/29/07