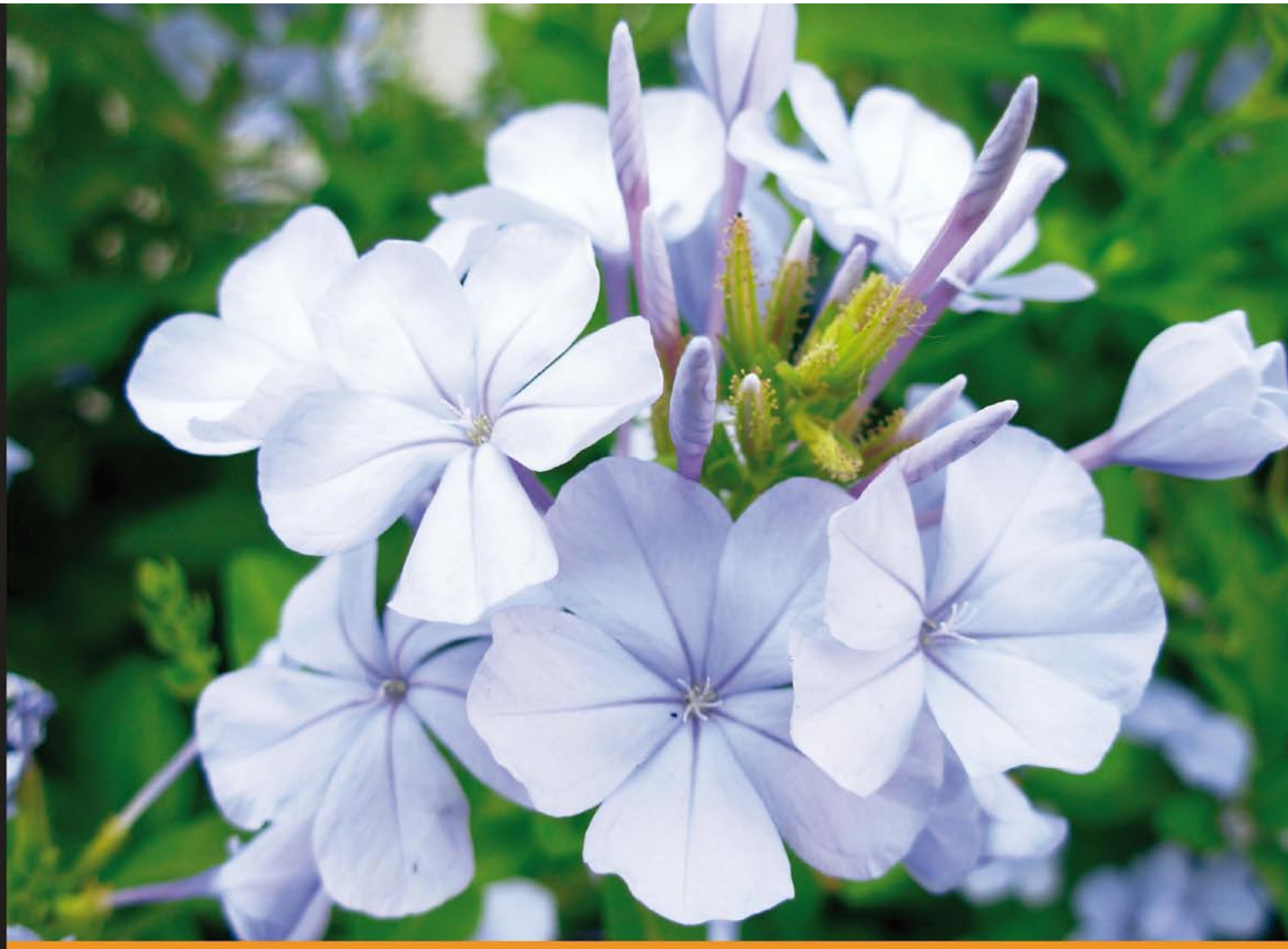


From Technologies to Solutions

Oracle Web Services Manager

Securing Your Web Services

Sitaraman Lakshminarayanan

PACKT PUBLISHING

Oracle Web Services Manager

Securing Your Web Services

Sitaraman Lakshminarayanan

PACKT
PUBLISHING
BIRMINGHAM - MUMBAI

Oracle Web Services Manager

Securing Your Web Services

First published: June 2008

Production Reference: 1020608

Published by Packt Publishing Ltd.
32 Lincoln Road
Olton
Birmingham, B27 6PA, UK.

ISBN 978-1-847193-83-4

www.packtpub.com

Cover Image by Nilesh R. Mohite (nilpreet2000@yahoo.co.in)

Credits

Author

Sitaraman Lakshminarayanan

Reviewers

Marc Chanliau

Rajesh Warrier

Acquisition Editor

Bansari Barot

Technical Editor

Usha Iyer

Editorial Team Leader

Mithil Kulkarni

Project Manager

Abhijeet Deobhakta

Project Coordinator

Lata Basantani

Indexer

Hemangini Bari

Proofreader

Cathy Cumberlidge

Production Coordinator

Shantanu Zagade

Cover Work

Shantanu Zagade

About the Author

Sitaraman Lakshminarayanan is an Enterprise Architect with over 11 years of IT experience in implementing software solutions based on Microsoft and Java platforms. His area of interest is in enterprise architecture, application integration and information security, and he specializes in identity and access management, web services and SOA. He is a co-author of ASP.NET Security (Wrox publications) and has presented at regional and international conferences on web services security and identity management.

I thank my wife, Vijaya for her exceptional support in finishing this book on time in the midst of her new job and a new addition to our family. I thank my mother for her constant love and support. I am grateful to the reviewers who provided valuable help to ensure content accuracy. I appreciate the help from the Packt Publishing team, especially Usha Iyer for reviewing and editing, Lata Basantani for coordinating the work between myself, the reviewers and the editorial team.

I dedicate this book to my late mother-in-law,
who wished me success in every step of my career since I met her.

About the Reviewers

Marc Chanliau has over 30 years' experience in the software industry including systems engineering, project and product management. Marc is currently responsible for the product management of platform security and web services security for Oracle's Fusion Middleware. Marc has been closely involved with XML standards development over the last 8 years, in particular SAML, WS-security, and WS-Policy. Marc holds an MS in Linguistics from the University of Paris (Jussieu).

I would like to thank all the developers and quality-assurance engineers of Oracle Web Services Manager for providing an amazing SOA and web services security tool that is being used by many customers worldwide.

Rajesh Warrier, currently working as one of the lead system architects in Emirates Group IT, has around 10 years, experience in the industry, working with companies like Sun Microsystems. Rajesh has been responsible for architecting and designing many mission-critical enterprise applications using cutting edge technologies, and is currently working as an architect and mentor for the new generation cargo system for Emirates airlines, developed completely using JEE. He has also reviewed another Packt book, Service Oriented Java Business Integration by Binildas C.A.

Table of Content

Preface

Oracle Web Services Manager, a component of SOA Suite from Oracle is a web services security and monitoring product that helps organizations not only to define and enforce security policies, but also to define and enforce the service level agreements. One of the key components of Service Oriented Architecture is security, and this book will be useful for those who are implementing SOA or for those who just want to manage and secure their web services.

This book not only describes the need for and the standards of web services security, but also how to implement them with Oracle WSM. It contains detailed examples on how to secure and monitor web services using Oracle WSM with explanations on the internals of WS-* security standards. It also describes how to customize Oracle WSM and how to plan for high availability.

What This Book Covers

Chapter 1 gives an in-depth overview of web services security from a business point of view, describing the security challenges in a web services environment, why traditional network security isn't enough, and how to measure the ROI on web services security.

Chapter 2 discusses the architecture of web services security including the various interoperable standards, challenges in implementing web services security in .NET and Java applications, and the need for centralized policy definition and enforcement. It also discusses the need to integrate with existing single sign-on systems and provides an overview of Oracle Web Services Manager.

Chapter 3 discusses the architecture of Oracle Web Services Manager. In this chapter, we explore the various components of Oracle WSM, such as gateway, agent, policy management, routing, monitoring, etc.

Chapter 4 talks in-depth about how to implement authentication and authorization in web services using Oracle WSM. It explains how to define security policy and protect web services with a detailed step-by-step example. Once you learn to authenticate and authorize web services requests, the next step is to protect the confidentiality of the message.

Chapter 5 discusses in-depth about encryption and decryption in web services and how to implement them using Oracle WSM with a detailed step-by-step example. This chapter also discusses how to test using a Microsoft .NET application and Oracle WSM test pages.

Chapter 6 addresses the most important part of web services security: digital signature. In this chapter, you will learn how to define security policy to digitally sign and verify SOAP messages with a detailed step-by-step example. This chapter also discusses how to test using Microsoft .NET application and Oracle WSM test pages.

Chapter 7 discusses the internals of Oracle WSM policy manager and how to implement a custom policy with an example scenario and a step-by-step description. No matter what features the Oracle WSM product offers, there may be reasons why you might want to implement certain custom security policies.

Chapter 8 discusses the deployment strategy, database options, high availability requirements and various options to deploy Oracle WSM. It is important that Oracle WSM is highly available to meet business needs.

Chapter 9 discusses the requirements to monitor the availability of Oracle WSM, how to define and monitor the service level agreements, performance metrics, etc.

Chapters 10 and *11* discuss the internals of XML encryption and XML signature standards and how they are used within WS-* security. We walk through with example SOAP messages and explain how encryption and signature are implemented.

Chapter 12 discusses how to combine both digital signature and encryption to ensure both confidentiality and integrity of the message. In this chapter, we will discuss how to implement sign and encrypt in Oracle WSM with a step-by-step example.

C*hapter 13* concludes the book with a discussion on Enterprise Security—web services and single sign-on and the need to bridge the gap between SSO products such as Oracle Access Manager and Oracle WSM with the introduction to security token service. We also discuss the integrated security architecture.

What You Need for This Book

You need Oracle Web Services Manager stand alone or the SOA Suite. This can be installed in Windows or Unix platform.

Who is This Book for

This book mainly targets developers, architects and technical managers with expertise in developing and deploying web services. The readers are expected to have a basic understanding of web services, and also development and deployment of web services.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

There are two styles for code. Code words in text are shown as follows: "We can include other contexts through the use of the `include` directive."

A block of code will be set as follows:

```
<?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">
```

Any command-line input and output is written as follows:

```
wsmadmin.bat start
```

New terms and **important words** are introduced in a bold-type font. Words that you see on the screen, in menus or dialog boxes for example, appear in our text like this: "clicking the **Next** button moves you to the next screen".

Reader Feedback

Feedback from our readers is always welcome. Let us know what you think about this book, what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply drop an email to `feedback@packtpub.com`, making sure to mention the book title in the subject of your message.

If there is a book that you need and would like to see us publish, please send us a note in the **SUGGEST A TITLE** form on `www.packtpub.com` or email `suggest@packtpub.com`.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on `www.packtpub.com/authors`.

Customer Support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the Example Code for the Book

Visit `http://www.packtpub.com/files/code/3834_Code.zip` to directly download the example code.

The downloadable files contain instructions on how to use them.

Errata

Although we have taken every care to ensure the accuracy of our contents, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in text or code—we would be grateful if you would report this to us. By doing this you can save other readers from frustration, and help to improve subsequent versions of this book. If you find any errata, report them by visiting `http://www.packtpub.com/support`, selecting your book, clicking on the **Submit Errata** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata are added to the list of existing errata. The existing errata can be viewed by selecting your title from `http://www.packtpub.com/support`.

Questions

You can contact us at `questions@packtpub.com` if you are having a problem with some aspect of the book, and we will do our best to address it.

1
Introduction to Web Services Security

Web services have become an integral part of software development and with the increased adoption of Service Oriented Architecture, business functionalities are being exposed as services within and outside the organization. While web services can expedite the integration process, business owners also need to understand the risk of Service Oriented Architecture from the security point of view and should make every attempt to mitigate that risk. This chapter will give an introduction to web services security—the need for it, what the security options are, and will even give a look quickly into the return on investment in web services security.

The Need for Web Services Security

Application integrations have gone through different phases from traditional file transfer to distributed systems, such as Distributed Component Object Model (DCOM) or Common Object Request Broker Architecture (CORBA) to the current platform independent standards based on Web Services (SOAP). Web services expedite the process of integrating different applications in different platforms without changing much of the underlying business process.

While web services can help the business, the data which was accessible only to that application is now available for integration with any other application. This leads to security concerns in exposing business functionality as web services. The most common questions asked by service providers and consumers in web services architecture regarding security are:

- Service provider:
    - How secure is the data being exchanged? How do I control access to the service?

    - How do I ensure the confidentiality and integrity of the message, while still maintaining the interoperability benefits of web services?
    - How do I manage the enhancements and fixes to the service without affecting the consumers?
    - How do I monitor the performance of the service, its availability, and so on? (various parameters of Service Level Agreement)

- Consumers:
    - How do I access multiple web services with their own unique security requirements?
    - How can I decouple the service invocation from the client application so that I can switch to a different service without affecting the client application?

There are some questions that are common to both service providers and service consumers, such as:

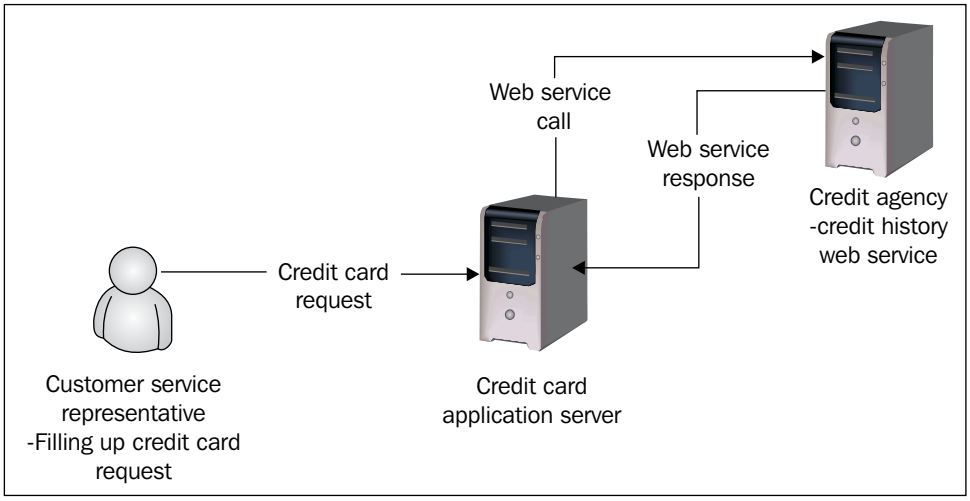
- How can I externalize the security so that I need not worry about various security implementations for the underlying business application or service?
- How can I define and enforce the governance around publishing web services, consuming service, security, and so on?

Security Challenges in a Web Services Environment

Web services actually expose the data and the business process without the need to have access to the user interface of that application. The security requirements in the web service environment can be best illustrated by an example.

Consider a scenario where a bank acquires information about its customer including a Social Security Number to open a credit card account. One critical requirement to open a credit card account is to have certain minimum credit history. The business process can be defined as:

- Obtain the customer's personal information, including Social Security Number.
- Perform credit check with an external agency via web service.
- Upon the approval of credit history, open the credit card account.



The picture shows the interaction between different systems. From the picture, it is clear that the security at the front end layer, that is at the application layer alone, is not enough.

When there is critical business information exchanged within web services, the service should:

- Identify the service consumer: Authentication
- Validate that the service consumer is authorized to access the service: Authorization
- Protect the message integrity: Signature
- Protect the confidential data: Encryption
- Track who accessed and when: Auditing

The Need for Identity Propagation from Calling Application to Web Services

Security at the application layer can show the pages for which the user has the privileges to view. It can even control the operations allowed on that web page, such as Add, Modify, Delete, and so on. This is typically done by means of a web access management product such as Oracle Access Manager. However, the application actually does certain operations on behalf of the user, such as accessing the database, accessing web services, and so on. Consider the scenario where the user at the travel department of your company makes travel reservations. The user will login to the

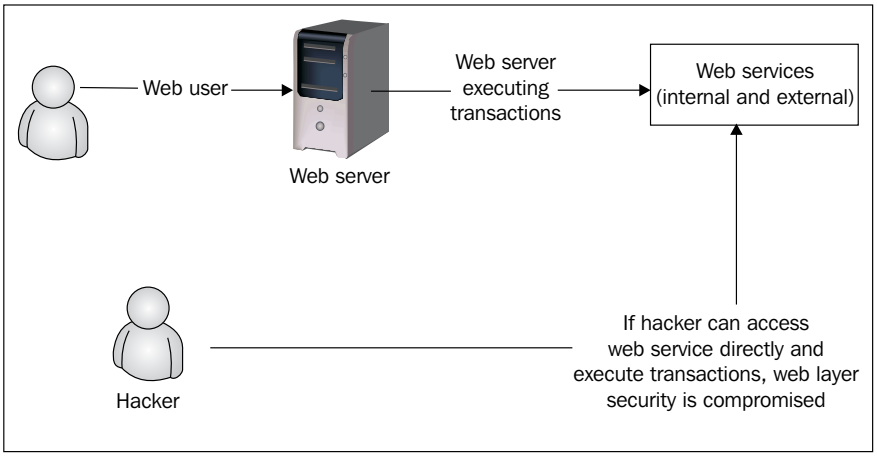
application and when the user actually makes the reservation, the application makes the web service calls to reserve the tickets and charge the credit card. In this scenario, the web application has the following capabilities (but not limited to):

- Accept credit card information
- Charge the credit card
- Make the reservation

In the above scenario, the application layer can only protect the information based on the user who logged in, and can only protect someone from accessing a web page that they are not supposed to. However, the application needs to call a few web services to complete the travel reservation. In that scenario, the challenges are:

- How do I propagate the identity of the logged-in user to the web services? How do I translate the SSO token that was established when the user logged into the system to one of the security tokens required by the web service provider?
- How do I add any additional information to the web service request so that the service provider can authenticate and authorize the request?
- How do I enforce security that may be different for different services without writing security-specific code within my application?

Since the web services are easily accessible, it is important to make sure that it is very well secured, not only to protect the confidentiality and integrity, but also to allow access only to the authorized users. The following figure shows the reason why web layer security is not enough for database application.

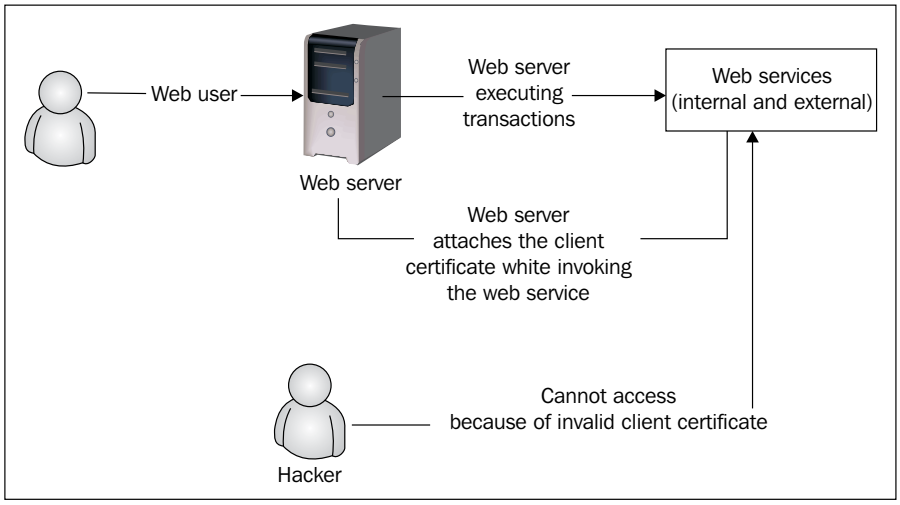


The next section looks at why network security isn't sufficient, and also provides an overview of different web service security components.

Why HTTPS Based Security Is Not Enough

One common security practice is to implement network level security, such as **HTTPS** for transport layer security. Secure Socket Layer (SSL) is the de-facto standard way of communicating over HTTP, commonly used as HTTPS, so that the data is encrypted over the network between the client (for example, web service consumer) and the web server (for example, web service provider). While HTTPS offers transport layer security by encrypting the data over the wire, it does not validate the user actually accessing the URL by default. HTTPS only assures the clients (consumers) that they are talking to the legitimate web site (by means of digital certificate). However, there is an option available to validate the client—by means of client certificate validation. In this case, the client application has to attach the client certificate while invoking the HTTPS URL, and the web server can validate the authenticity of the client certificate before actually granting access to the URL. HTTPS with the client certificate validation will ensure that the server knows exactly who the consumer is.

In our example application scenario, one easy way to enforce secure access to the web service is to enable HTTPS along with client certificate validation between the web service provider and the web service consumer. The following diagram shows how the web service is given access only to a trusted organization, and the intruder is prevented from accessing the web service.



Certain network level security such as HTTPS, client certificate validation can be applied even to web services. However, each has its own pros and cons, and a detailed discussion is beyond the scope of this book. We will describe the network level security for web services in depth here:

Network level security	Description	What it doesn't do for web services
HTTPS (SSL)	It creates a secure connection with server and thus encrypts the data transmitted over the network.	HTTPS access alone does not validate the client/consumer of the web service.
HTTPS (SSL) with Mutual Authentication	It encrypts the data over the network after performing client certificate validation.	Even though it can encrypt and identify the client or the consumer, it does not ensure the integrity of the message that actually left the server.

While network level security is a great place to start, for web services to secure the transaction at the network layer, the data or the message itself should be protected once the message leaves or reaches the destination. The next section will provide an overview of the different components of web services security, and why it is important to address them.

Components of Web Services Security

The various components of web services security are:

Authentication

Authentication is the process of verifying the credentials of the user accessing the system. Webservices should authenticate the user requesting the service before it can execute the request. Without any proper identification of the user, it will be impossible to perform an audit of any unauthorized access.

Authorization

Authorization is the process of validating who has access to what before granting access to perform an operation. Web services security should not only authenticate the user, but should also validate if the user has enough privileges to perform the operation.

Confidentiality

Web services exchange critical business information in XML; certain information can be confidential and it should be protected from any intruder or unauthorized access. XML messages should be encrypted by the service, or the consumer, depending on the sensitivity of the message being exchanged.

Integrity

Since web services exchange XML messages, the integrity of the message should be ensured. Message integrity can be ensured by means of digital signature. Digitally signing the XML messages in the web services can ensure that the message was not tampered by the intruder in transit.

Return on Investment

Investment in security is not easily quantifiable as any other IT system investment. While there is a technology component to the security implementation, it is often the risk that the business is willing to take that drives the security implementation, and securing the web services is no different.

Implementing web services security does require certain investment, either in terms of buying an off-the-shelf product such as Oracle Web Service Manager, or implementing a custom security framework across all the web services and the clients. In either case, the investment should be justified for the business owners. While calculating the ROI on web services security implementation, the following factors should be considered from the business stand point:

- How much of the confidential data is being exposed on services?
- Is it fine not to have any data integrity checks on those web service messages?
- What would be the business impact? Lost monetory value, lost productivity, reputation, and so on.
- Should the service authenticate the user? Is it okay for anyone to access this service?
- Is it okay for everyone to perform all the operations exposed by the service?
- Is it required to satisfy any government regulations?

If the business owners decide to make sure that the web services should be secured, then the IT department should consider the following while defining the ROI:

- How many web services are deployed in a single server, or multiple servers?
- Is there a potential for many more services in the future?
- Are those services in different platforms such as Microsoft .NET and Java?
- Are there any web services behind the firewall that need to be exposed to external vendors?

When there are many web services in different platforms, and that require the understanding of multiple authentication tokens, encryption, signature and so on, the cost of custom development and maintenance of security should be considered and compared with the cost of the product and its performance benefits.

Summary

This chapter gives an introduction to why web services should be secured in an organization and explains the different components of web services security that should be addressed. The next chapter will describe the various standards in web services security, the importance of centralized policy manager, and will also explain the web services security from an architect's point of view.

2
Web Services Security—Architectural Overview

Integrating applications and business processes across various platforms is made easier with web services and it is important that the web services security be considered as an integral part of the organization's web services initiative. As the organization evolves and exposes or integrates with many web services, the management of security policies and enforcement across all the applications can get cumbersome. One way to overcome the challenge is to externalize the security to another application such as Oracle Web Services Manager. In this chapter, we will discuss the need for centralized management of web services, policy definition and policy enforcement with a quick introduction to Oracle Web Services Manager.

Overview of XML Security Standards

Web services is nothing but a set of **XML** messages that are exchanged between software applications in various platforms to provide a better integration platform for a distributed environment. Since its adoption to facilitate faster integration of disparate business processes, requirement for interoperable security definitions has become imminent. In this section, we will take a closer look at why existing security implementations such as SSL are not sufficient, and the need for standards such as **WS-Security**.

Closer Look at SOAP Messages

Consider the example of an online ecommerce web site where you pay for the shopping using credit or debit card. The online ecommerce web site leverages web services to talk to the payment gateway to process the transactions. A typical SOAP message under this scenario for charging a credit card would look like:

```
<?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:Body>
    <ChargeToCreditCard xmlns="http://tempuri.org/">
      <CreditCardNumber>309192281120999</ CreditCardNumber >
      <Amount>10</ Amount >
    </ ChargeToCreditCard >
  </soap:Body>
</soap:Envelope>
```

The SOAP message above consists of a SOAP envelope that includes a SOAP header and a SOAP body. The SOAP body contains the actual message that the SOAP header is provided with to add any additional processing information that the application can extend to. In the above example, SOAP body contains the information to charge the credit card and the SOAP header does not have any additional information.

The above payment gateway web service, if invoked as such without SSL, causes the SOAP message described to be actually visible for anyone who can sniff the network traffic. Even with SSL, the content is accessible as there is no authentication or authorization of the request. In order to effectively protect the customer information and the business, the web service should:

- Authenticate the user
- Encrypt the SOAP body message
- Digitally sign the SOAP body message

Let's take a closer look at each one of these security measures and explore how it can be done without any interoperability requirements. Then we will describe the need for WS-security standards.

Authentication

Authentication is the process of validating the credentials such as username and password. In the previous example, if the payment gateway web service were to be protected with a username and password, the client, i.e. the ecommerce web site, has to send the username and password in the web service request. Now the question arises as to how the username and password is to be sent in the web service SOAP request. One possible option is:

- Payment gateway web service can request username and password in the SOAP header.
- Client application, i.e. the ecommerce web site, has to construct the SOAP header with username and password and attach it to SOAP envelope/web service request.
- Payment gateway will parse the SOAP header and validate the username and password.

The sample SOAP message with username and password as a custom SOAP header would look like:

```
<?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>
    <CustomSoapHeader xmlns="http://tempuri.org/">
      <UserID>eCommerceClient</UserID>
      <Password>electronics</Password>
    </CustomSoapHeader>
  </soap:Header>
  <soap:Body>
    <ChargeToCreditCard xmlns="http://tempuri.org/">
      <CreditCardNumber>309192281120999</ CreditCardNumber >
      <Amount>10</ Amount >
    </ ChargeToCreditCard >
  </soap:Body>
</soap:Envelope>
```

In the example SOAP message, the payment gateway can only accept username and password, and validate if (and only if) specified in the specified SOAP header. The drawbacks of sending username and password in a custom SOAP header are:

- Each service provider will come up with their own custom SOAP header with different names, such as uid, username, password, etc.

- The service provider has to communicate various token types for each service.
- Applications that are accessing multiple services have to deal with attaching credentials in different formats for different services.
- Maintenance of both service and the client application becomes unmanageable when there are multiple services.

The above drawbacks make a good case for creating interoperable standards for describing authentication information. WS-security standards have incorporated several token profiles to support a wide variety of authentication mechanisms, such as username and password, Kerbores, SAML, etc.

Let us assume that the payment gateway web service is modified to accept username and password as per the WS-security standards; then the SOAP message would look like:

```
<?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
xmlns:S11="http://docs.oasis-open.org/wss/oasis-wss-wssecurity-secext-
1.1.xsd ">
   <S11:Header>
         <wsse:Security>
               <wsse:UsernameToken>
                     <wsse:Username>eCommerceClient</wsse:Username>
                     <wsse:Password>electronics</wsse:Password>
               </wsse:UsernameToken>
         </wsse:Security>
   </S11:Header>
  <soap:Body>
    <ChargeToCreditCard xmlns="http://tempuri.org/">
      <CreditCardNumber>309192281120999</ CreditCardNumber >
      <Amount>10</ Amount >
    </ ChargeToCreditCard >
  </soap:Body>
</soap:Envelope>
```

When service providers can accept the username and password, as per WS-Security standards, it becomes easier for both providers and consumers to implement the security in a standard, as opposed to custom implementations that are specific to a service provider or consumer.

In the next section we will explore the importance of WS-security standards for exchanging encryption information.

Confidentiality

The SOAP message that the payment gateway web service receives with the credit card information should be encrypted to protect the customer information. The SOAP message without encryption would look like:

```
<?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
xmlns:S11="http://docs.oasis-open.org/wss/oasis-wss-wssecurity-secext-
1.1.xsd ">
    <S11:Header>
          <wsse:Security>
                <wsse:UsernameToken>
                      <wsse:Username>eCommerceClient</wsse:Username>
                      <wsse:Password>electronics</wsse:Password>
                </wsse:UsernameToken>
          </wsse:Security>
    </S11:Header>
  <soap:Body>
    <ChargeToCreditCard xmlns="http://tempuri.org/">
      <CreditCardNumber>309192281120999</ CreditCardNumber >
      <Amount>10</ Amount >
    </ ChargeToCreditCard >
  </soap:Body>
</soap:Envelope>
```

The traditional way of encrypting is to use a shared secret key. In this case, the secret key is shared between the service provider and the consumer, and the information is encrypted. Let's consider the case where the payment gateway web service requires the credit card number to be encrypted using Triple DES algorithm and the encryption key is already shared between the consumer and the service provider. The SOAP message with an encrypted credit card number will look like:

```
<?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
xmlns:S11="http://docs.oasis-open.org/wss/oasis-wss-wssecurity-secext-
1.1.xsd ">
    <S11:Header>
          <wsse:Security>
                <wsse:UsernameToken>
                      <wsse:Username>eCommerceClient</wsse:Username>
                      <wsse:Password>electronics</wsse:Password>
```

```
                    </wsse:UsernameToken>
                </wsse:Security>
            </S11:Header>
            <soap:Body>
              <ChargeToCreditCard xmlns="http://tempuri.org/">
                <CreditCardNumber>AEB123XY231ENCRYPTED</ CreditCardNumber >
                <Amount>10</ Amount >
              </ ChargeToCreditCard >
            </soap:Body>
        </soap:Envelope>
```

In the above example, we only talk about exchanging credit card numbers and the SOAP message looks pretty simple. But in reality, the XML data that are exchanged are far more complicated as it has to describe necessary information for that business transaction.

In the above SOAP message, the service provider makes the following assumptions:

- The data in `CreditCardNumber` XML element is an encrypted value.
- The encryption algorithm is Triple DES.
- The encryption key is the one that is exchanged between the service provider and the consumer.

The challenges associated with the above approach for sending encryption information are:

- The service provider has to explain which data elements are encrypted and the algorithm, key, etc. too.
- The service provider has to give a new key for each consumer.
- The service provider has to differentiate the SOAP messages for each consumer by means of some identifier for using different encryption keys.
- The challenges are the same when service encrypts the response message and the consumer has to decrypt the information.

The challenges are the same or more complicated even when a different algorithm such as Public Key–Private Key is used to encrypt data in web services.

Since it is all XML data that is exchanged in web services, the recipient of the encrypted message has to know the basic information about encrypted data, such as:

- What data (i.e. XML) elements are encrypted?
- What is the algorithm used to encrypt the data?

- Is encryption based on shared secret (symmetric) or public-private key (asymmetric)?
- Where is the encryption key if it is exchanged along with the data?

WS-security standards adopted the XML encryption standards from W3C to represent the encrypted data information. The SOAP message below describes that the `CreditCard` element data is encrypted using Triple DES as per WS-security/ XML encryption standards.

```
<?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
xmlns:S11="http://docs.oasis-open.org/wss/oasis-wss-wssecurity-secext-
1.1.xsd ">
    <S11:Header>
          <wsse:Security>
                <wsse:UsernameToken>
                      <wsse:Username>eCommerceClient</wsse:Username>
                      <wsse:Password>electronics</wsse:Password>
                </wsse:UsernameToken>
          </wsse:Security>
    </S11:Header>
  <soap:Body>
    <ChargeToCreditCard xmlns="http://tempuri.org/">
    <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#"
Type="http://www.w3.org/2001/04/xmlenc#Element" ID="ED">
      <EncryptionMethod Algorithm="http://www.w3.org/2001/04/
xmlenc#tripledes-cbc" />
      <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:KeyName>Test</ds:KeyName>
      </ds:KeyInfo>
      <CipherData xmlns="http://www.w3.org/2001/04/xmlenc#">
        <CipherValue> 0qn/Au2bWIZNJsM3I/7nkk0Tv0OSUbh8c=</CipherValue>
      </CipherData>
    </EncryptedData>
      <Amount>10</ Amount >
    </ ChargeToCreditCard >
  </soap:Body>
</soap:Envelope>
```

In the SOAP message above shown the `EncryptionMethod` clearly explains that it is a Triple DES algorithm and that `KeyName` is an identifier to get the key name. One should also note that the `EncryptedData` element has an attribute called `Type` that shows that the data that was encrypted was an XML element. What that means is, you can also encrypt the entire XML document, i.e. the entire SOAP message, if required. This flexibility is mainly there to address any performance constraints that we might have on large SOAP messages. (Encryption is a fairly expensive CPU intensive task.)

The above SOAP message is an example for representing encryption information in an interoperable manner and it also demonstrates that WS-security addresses by adopting XML encryption standards from W3C. In the next section, we will explore the need for interoperable standards to ensure the integrity of the message.

Integrity

Authenticating the consumer ensures that a valid user is accessing the service and encrypting the information will keep it confidential. However, the recipient of the message should ensure that the message was not modified by anyone or replaced with a different encrypted data.

A SOAP message that was sent to the payment gateway with the credit card Number encrypted using the public key of the payment provider, would look like:

```
<?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
xmlns:S11="http://docs.oasis-open.org/wss/oasis-wss-wssecurity-secext-
1.1.xsd ">
  <soap:Body>
    <ChargeToCreditCard xmlns="http://tempuri.org/">
    <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#"
              Type="http://www.w3.org/2001/04/xmlenc#Element" ID="ED">
      <EncryptionMethod Algorithm=
             "http://www.w3.org/2001/04/xmlenc#RSA" />
      <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:KeyName>Test</ds:KeyName>
      </ds:KeyInfo>
      <CipherData xmlns="http://www.w3.org/2001/04/xmlenc#">
        <CipherValue> 0qn/Au2bWIZNJsM3I/7nkk0Tv0OSUbh8c=</CipherValue>
      </CipherData>
    </EncryptedData>
      <Amount>10</ Amount >
    </ ChargeToCreditCard >
  </soap:Body>
</soap:Envelope>
```

Since the above message is encrypted using the public key of the payment gateway, anyone can get access to the same public key (if it's made public so that other business partners can access it easily) and can send the encrypted message.

The SOAP message below is also encrypted with the same public key and even the same credit card information is encrypted. So any intruder can actually replace the encrypted portion of the message and replace the credit card number.

In the SOAP message below, the `CipherValue` is actually different, which could map to any other valid credit card number.

```
<?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
xmlns:S11="http://docs.oasis-open.org/wss/oasis-wss-wssecurity-secext-
1.1.xsd ">
  <soap:Body>
    <ChargeToCreditCard xmlns="http://tempuri.org/">
    <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#"
              Type="http://www.w3.org/2001/04/xmlenc#Element" ID="ED">
      <EncryptionMethod Algorithm=
              "http://www.w3.org/2001/04/xmlenc#RSA" />
      <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:KeyName>Test</ds:KeyName>
      </ds:KeyInfo>
      <CipherData xmlns="http://www.w3.org/2001/04/xmlenc#">
        <CipherValue> afaf3234234kjkadfa==</CipherValue>
      </CipherData>
    </EncryptedData>
      <Amount>10</ Amount >
    </ ChargeToCreditCard >
  </soap:Body>
</soap:Envelope>
```

Payment gateway has to ensure that the message was not tampered with or altered by anyone, and hence requires digital signatures to sign the message before it is actually encrypted.

The way digital signatures work is by first creating a one way hash (digest) of the message and then encrypting the hash value using the private key of the sender. The recipient will recalculate the hash value and encrypt using the public key to compare the value.

In the payment gateway example, in order to protect the integrity of the message, the credit card number and the amount should be digitally signed first and then encrypted to protect the confidentiality of the message. Now the signed and encrypted message would look like:

```
<?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
xmlns:S11="http://docs.oasis-open.org/wss/oasis-wss-wssecurity-secext-
1.1.xsd ">
  <soap:Body>
    <ChargeToCreditCard xmlns="http://tempuri.org/">
    <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#"
              Type="http://www.w3.org/2001/04/xmlenc#Element" ID="ED">
      <EncryptionMethod Algorithm=
              "http://www.w3.org/2001/04/xmlenc#RSA" />
      <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:KeyName>Test</ds:KeyName>
      </ds:KeyInfo>
      <CipherData xmlns="http://www.w3.org/2001/04/xmlenc#">
        <CipherValue> 0qn/Au2bWIZNJsM3I/7nkk0Tv0OSUbh8c=</CipherValue>
      </CipherData>
    </EncryptedData>
      <Amount>10</ Amount >
    </ ChargeToCreditCard >
    <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
      <SignedInfo>
        <CanonicalizationMethod Algorithm=
                  "http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />
        <SignatureMethod Algorithm=
                  "http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
        <Reference URI="#xpointer(/)">
          <Transforms>
            <Transform Algorithm="http://www.w3.org/2000/09/
                  xmldsig#enveloped-signature" />
          </Transforms>
          <DigestMethod Algorithm=
                  "http://www.w3.org/2000/09/xmldsig#sha1" />
          <DigestValue>AZ/q/WCsODcM0tfcoGzGgraVxxk=</DigestValue>
        </Reference>
      </SignedInfo>
<SignatureValue> gv0s0cFlboSSbF/PlnMQw9ygH6+E6msSX8c=</SignatureValue>
      <KeyInfo>
        <KeyValue xmlns="http://www.w3.org/2000/09/xmldsig#">
```

```
            <RSAKeyValue>
<Modulus>fFnoomEnyk10=</Modulus>
              <Exponent>AQAB</Exponent>
            </RSAKeyValue>
          </KeyValue>
        </KeyInfo>
      </Signature>
    </soap:Body>
</soap:Envelope>
```

In the SOAP message above, the entire SOAP body is digitally signed and then the credit card number information is encrypted. The `DigestValue` will not match when the data is tampered with or replaced with other encrypted data.

In the SOAP message above, the most commonly asked questions about digital signatures are:

- What is the hash algorithm used?
- What is the hash value?
- What data is hashed?
- What is the information regarding the public key?

The XML signature specification from W3C addresses the interoperability concerns around exchanging signed XML messages, thus addressing the above mentioned questions. Based on the SOAP message above, it is clear that the hash algorithm used is "`sha1`", the hash value is represented by the `DigestValue` element and that the `KeyValue` element describes the public key. The XML signature standards from W3C are now incorporated in WS-security to ensure the integrity of the message.

Overview of WS-Security Standards

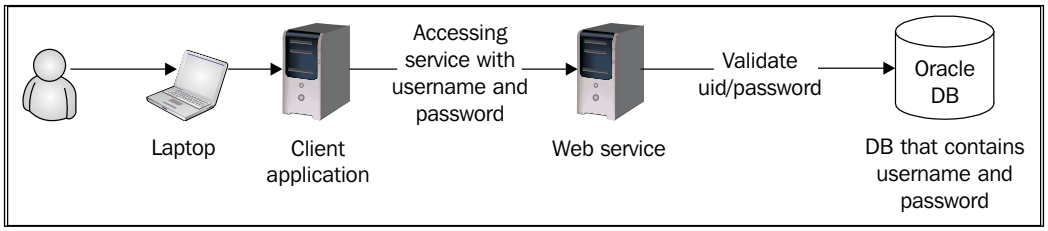
In the last section, we explored how the XML encryption and XML signature address the interoperability requirements around exchanging encrypted or digitally-signed messages. In web services, all the messages are XML messages. Hence, the WS-security standards from Oasis addresses the encryption and signature requirements by incorporating the XML encryption and XML signature specifications from W3C. WS-security standards also include security token profiles, such as username, Kerberos, SAML, and X.509 to address the need for security token (authentication and authorization). In this section, we will take a closer look at WS-*security standards.

Implementing WS-*Security in Applications

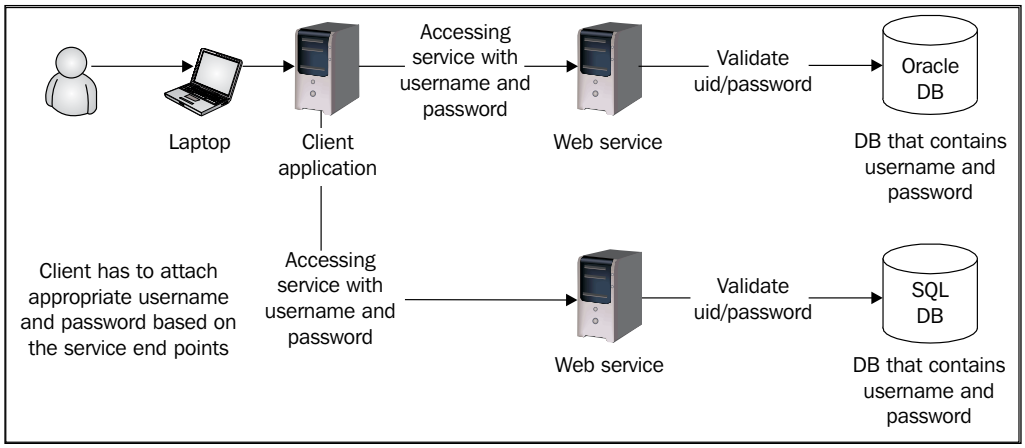
WS-security standards are a way to represent security information. Each application has to leverage its native API or external product to actually implement the WS-security standards.

Consider for example, a web service that requires the client to authenticate using a username and password. In this case, the web service will receive the SOAP message with the username and password. The web service has to perform the actual validation of username and password against a database, or LDAP directory, or other data store. The detailed steps to perform the authentication are:

- Service will accept the SOAP message.
- Service will parse the SOAP message to obtain the username and password.
- Service will validate the username and password against a data store such as LDAP or RDBMS.



Now let's consider the case where the client application has to talk to two different web services, each requiring different usernames and passwords to authenticate to the service. The following figure shows the client and service interaction.



In the above diagram, the client application has to perform the following actions:

- Identify which service end point is being invoked.
- Look up the username and password from a configuration file or database.
- Attach the username and password to the appropriate web service.
- Invoke the web service.

The complexity grows as the client application talks to many services that require different usernames and passwords to authenticate. On the service provider side, the complexity increases with multiple web services using different databases for authenticating credentials. It also increases the security risk by not maintaining credentials in one place, and the cost of enforcing security best practice increases due to duplicating efforts across multiple services and databases.

If you thought that authentication itself is complicating your web services security implementation, think about implementing encryption or digital signatures.

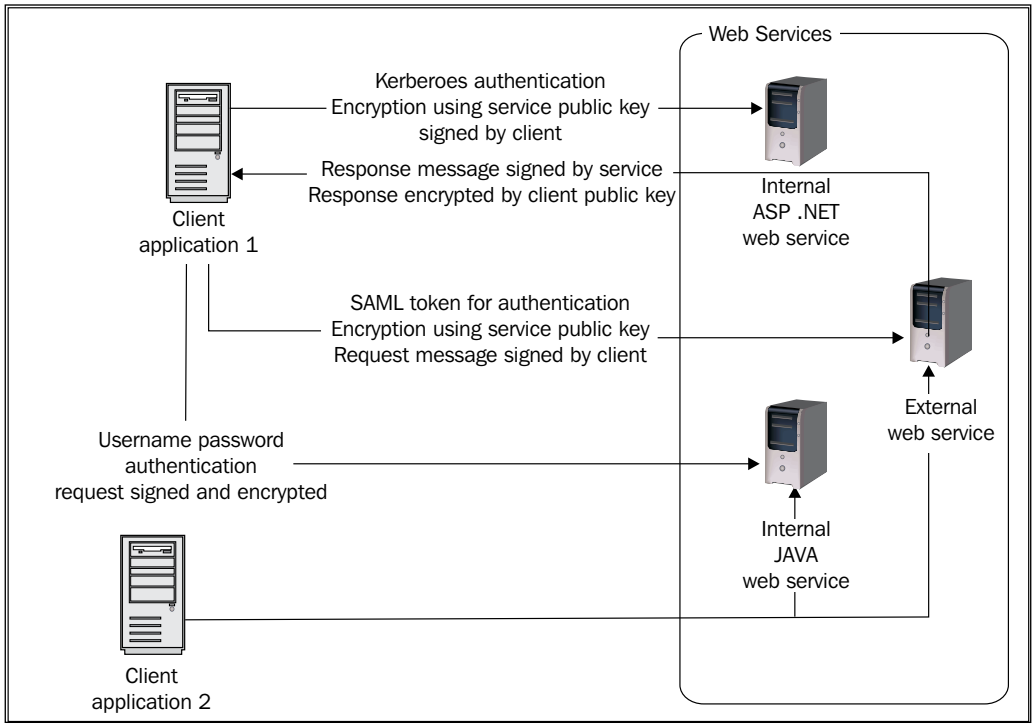
Consider the example of payment gateway service provider that requires the incoming message to be encrypted using the public key of the payment gateway. Web service has to perform the following operations:

1. Look up the private key based on a certain key identifier
2. Decrypt the message.

Now if the web service is encrypting the response message back to the consumer, it has to encrypt with the public key of the client application. Now the service provider has to perform the following operations:

- Look up the public key based on some identifier.
- Encrypt the message with appropriate public key.

Consider the following diagram where you have two internal web services and one external web service being accessed by one or more applications.



In the diagram above, the two internal web services have to:

- Enforce two types of authentication—**username/password** and **Kerberos.**
- Verify digital signatures from two different clients.
- Encrypt response message from one of the web services.

With one **external web service** each client has to:

- Be accessed from two different clients.
- Implement the security policy, such as service public key certificates.
- Deal with any changes in security policy.

Both the service provider and the consumer, which are business applications, now have to deal with a lot of security implementation, and that creates a challenge in terms of design, development, deployment, and maintenance.

Now imagine having multiple client applications having to deal with multiple services. Both the client and the service has to deal with the same challenge of enforcing the security policy. This leads to the need for centralized management of web services security, which is discussed in the next section.

Centralized Management of WS-*Security

One of the main reasons for the faster adoption of web services is its support for interoperability and ease of integration, without worrying about firewall or sticking to one particular technology or standard. Security should not be a hindrance to the adoption of services, even with standards such as WS-security. In this section, we will discuss the need for centralizing the web services security operations and the benefits of centralizing them.

The Need for Centralizing WS-*Security Operations

The service providers (web service providers) should fully leverage the WS-security standards and should be able to accept and process multiple authentication tokens, different encryption algorithms, flexibility in what data is signed and what data is encrypted, etc. The service consumers also should send the necessary security tokens and perform the required encryption and signature, as required by the service provider.

The implementations of WS-security components, such as encryption, signature, Kerberos token, username token, etc. are available in both .NET and Java platform. However, the developers have to spend time configuring security within each application they write, and it becomes an administrative nightmare when it comes to managing web services security across a variety of applications in various platforms. The complexity increases when SSL is required for a certain web service (providers or consumers).

In order to perform all these operations, each web service has to maintain multiple public keys and private keys all over the organization. It becomes unmanageable when a key is lost, stolen or expired and needs to be updated across all the servers. Based on the challenges described, there is a strong need to centralize the WS-security operations in one place (e.g. Oracle Web Services Manager).

Benefits of Centralizing Web Services Security Operations

While it is apparent that the web services security operations should be centralized, there are more benefits in centralizing the security operations:

- Developers need not implement security in each web service or client application.
- Administrators can define the security policy, i.e. what data should be encrypted, signed, what type of authentication tokens are accepted, etc.
- Security operations are moved away from the core service or consumer application.
- It draws an inherent benefit of increasing the performance by moving the security to another process or server.

In order to centralize the web service security operation, organization, should either custom implement the security operations or adopta web services security product, such as Oracle Web Services Manager. In the next section, we will briefly talk about this product.

Introduction to Oracle Web Services Manager

In the last section we explored the importance of centralizing the web services security operations. However, any custom implementation would be too expensive to develop and maintain, especially with the fast-growing changes in the web services standards and WS security standards. In this section, we will take a took at the Oracle Web Services Manager product to secure the web services.

Oracle Web Services Manager can actually centralize the security operation by merely acting as a gateway for all your services, and the Oracle WSM product will perform all the security operations. Oracle WSM brings the benefit of:

- Centralizing security operations
- Defining different policies for different web services
- Creating policy templates that can be applied over a set of services
- Enhanced logging capability for troubleshooting
- Flexible auditing mechanism to track the performance
- Strong integration with Oracle Access Manager and Siteminder
- Flexible architecture for easy customization
- Monitor web service operations, security operations, etc.
- Manage the web services deployment
- Generate reports about web services performance, usage, etc.

Oracle Web Services Manager can actually reduce the security development effort and can offload the security processing to another server.

Summary

In a service oriented architecture, security should be included right from the design, and the services should support all the WS-security standards, and should be flexible enough to adapt to any new standards. The security architecture should be flexible enough that all the security operations can be offloaded to another process with the possibility of customizations to support any specific requirements. Products such as Oracle Web Services Manager can help implement the web services security and centralize the policy for web services with the flexibility for customization to support any custom security requirements. In the next chapter, we will take a closer look at the architecture of Oracle Web Services Manager.

3
Architecture Overview of Oracle WSM

The service oriented architecture using web services has a strong need to secure the web services and to monitor the web services infrastructure. The security infrastructure for web services should address authentication, authorization, confidentiality, integrity and non-repudiation, and should also support the interoperability standards. The SOA (using web services) infrastructure should also have the ability to monitor the availability of services to provide timely and on-demand reports about the success and failure of various service operations. Oracle Web Services Manager is the web services security and monitoring product from Oracle that addresses both the security and monitoring aspect of the web services infrastructure. Oracle WSM supports WS-security standards to effectively secure the web services, and the product itself is designed with greater flexibility to support any new standard or even write custom extensions. In the first two chapters, we talked about the need for web services security and in this chapter, we will take a closer look at the architecture of Oracle Web Services Manager.

Oracle WSM Architecture

In a service-oriented architecture using web services, organizations can expose their service offerings as web services and their business partners can consume their services to create business solutions. A good example would be a HR System which can integrate with a background check service provided by an external organization to perform a background check before an applicant is hired or offered a job.

In the previous example, web services are exposed by the background check service provider and the service is consumed by many organizations (HR systems of various organizations). In order to effectively maintain the defined service level agreements and also to ensure that only authorized organizations can invoke the service, and still ensure confidentiality and integrity of the data, the service provider exposing web services should:

- Externalize the security implementation.
- Centralize the security definitions or policies, i.e. what type of authentication tokens are accepted, what data elements are encrypted, etc.
- Change the security definition without having to redeploy any code or not affecting the actual service implementation.
- Integrate with existing infrastructure such as LDAP compliant directory or access management products.
- Monitor the web service for performance and availability.
- Expose service to business partners outside the firewall.

On the other hand, the service consumers should also have the flexibility to:

- Adhere to the appropriate security policy of the service provider.
- Externalize the security implementation.
- Flexible enough to Change security implementation or adapt to the service provider's new security policies.

Oracle Web Services Manager addresses the above mentioned requirement, and the architecture of Oracle WSM consists of:

- Oracle WSM Gateway: proxy-based approach to separate the security enforcement from the web service.
- Oracle WSM Policy Manager: Policy manager to define the web services policy.
- Oracle WSM Server Agents: Enforces security policy at the service end.
- Oracle WSM Client Agents: Adheres to the service policy defined by attaching appropriate tokens or performing encryption or signature.
- Oracle WSM Monitor: To monitor the performance, service level agreements, etc.

Oracle WSM Policy Manager

Securing a web service involves one of the many steps, such as authentication, authorization, encryption, decryption, signature generation, signature verification, etc. It is easy to manage when the security is not complicated and there is only a handful of services. It becomes unmanageable when the number of services increases or the complexity of the security, such as multiple authentication token, verifying incoming signature vs signing outgoing message, etc. increases. The Policy Manager component of Oracle WSM not only centralizes the security policy administration, but also makes it easy to attach existing policies to new services.

Overview of Oracle WSM Policy Manager

The web services security has various components, such as authentication, authorization, confidentiality, integrity and non-repudiation. Each of these components can have multiple ways to perform the operation, i.e. authentication can happen based on username and password, or based on SAML token, etc. Let's take a brief look at individual components and then describe how Oracle WSM can help in addressing the security requirements.

Authentication

By definition, it is a process of validating the user's credentials, i.e. who they say they are matches with who they are. The credentials can either be what they know, such as username and password, or what they have, such as a digital certificate issued by a trusted authority, or what they are, i.e. biometric information.

In the context of web services, the service provider should be able to authenticate the consumer based on the information that is presented as a part of the web service message. The information can be username and password, or a SAML token, or another SSO token or any other custom or other token type. Whatever the information is, it needs to be extracted and validated.

The service provider should be able to support multiple token types for authentication, and should also be able to integrate with various data stores within the organization, such as the LDAP compliant directory or another access management system to validate the credentials.

Authorization

Authorization or access control is the means of granting access to a specific resource (such as invoking the web service) for the authenticated user. The authenticated user should have the necessary privileges to perform the requested action.

In the web services world, the service provider should be able to validate that the authenticated user has the privileges to invoke the service. It is usually done either by validating whether the user belongs to a particular role or by validating the user's attributes, such as title that has a value of "Hiring Manager".

Confidentiality

Confidentiality is nothing but ensuring the privacy. In the web services world, the message should be encrypted so that no one other than the intended recipient can see the message.

The web service provider or consumer should be able to handle both encryption and decryption, i.e:

- Decrypt incoming SOAP message for certain web services
- Encrypt the response message

Integrity and Non-Repudiation

In the web service world it is important to ensure that the message is not altered during the transit, and that can be accomplished by digitally signing the message. Digital signature also creates a unique time stamp and validates the sender, thus providing a mechanism to establish non-repudiation. The web service provider or consumer should be able to:

- Validate incoming SOAP message signatures for certain web services.
- Sign a response message for certain web services.
- Sign and encrypt certain response messages.

Web services security requirements include token translation, i.e. converting one authentication token to another authentication token, and transport level security such as SSL too.

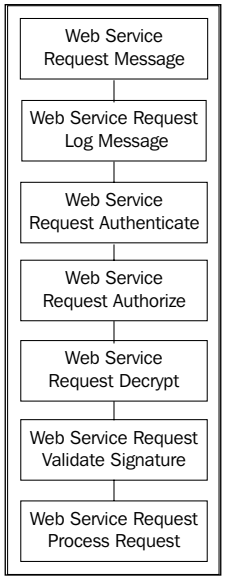
A heavily used web services architecture would require a flexible security management product that can externalize the security implementation with a flexibility to support these requirements, and any other custom requirements. Oracle WSM can let you define the security policies and attach it to web services. It also provides a flexible architecture that supports various token formats, authentication stores and is even customizable to meet any unique security need.

Oracle WSM Policy Manager consists of pipeline templates which are made up of different policy steps. Two key pipeline templates are Request Pipeline and Response Pipeline. Each of them consists of its own policy steps to process the request and response message.

Note: PreRequest and PostRequest pipleline templates will be removed in the future version of Oracle WSM.

Policy Steps and Pipeline Templates

Oracle WSM Policy Manager comprises Request and Response pipeline templates, which in turn consist of a set of policy steps. Policy steps are nothing but individual operations, such as authentication, authorization, encryption, and signature. With reference to the use case described in the earlier section, the various policy steps are as shown in the next figure:



The previous diagram shows the Request pipeline template with the policy steps.

The policy steps described are generic policy steps. However, in actual implementation, the authentication can be against any LDAP compliant directory, and the authorization can be against any LDAP compliant directory or custom data store.

Consider for instance, two web services exposed by consumer electronics corporation and each of them require authentication against the file system. All the response messages are digitally signed. To summarize, the requirements are:

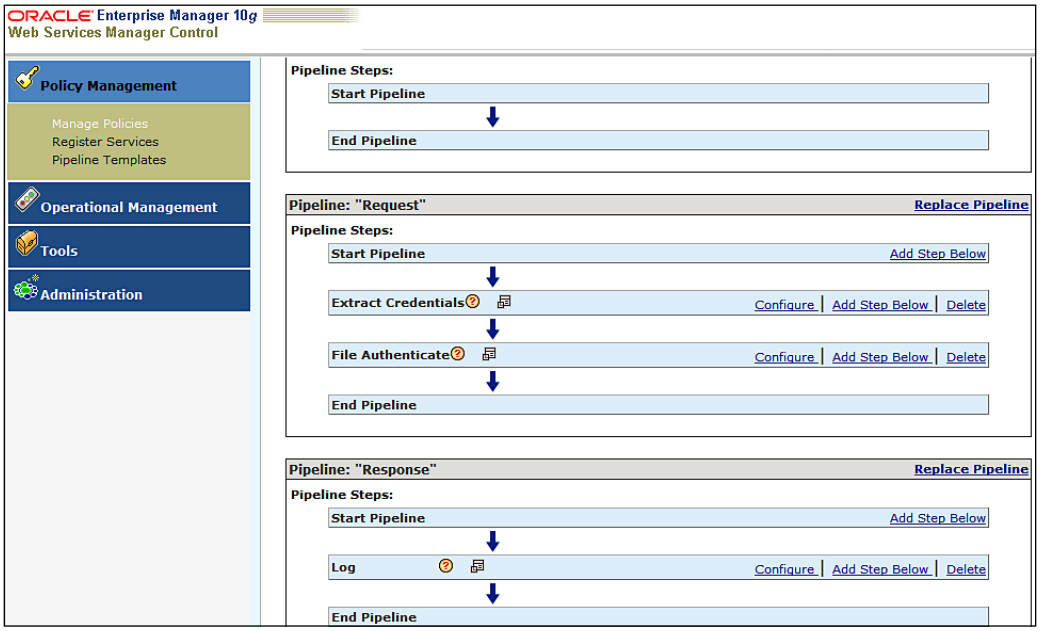
- Two web services exposed by the organization
- Two web services require authentication against file system
- Response messages are digitally signed
- Web services security is enforced by Oracle WSM

There are a couple of ways in which the policy can be defined and enforced by Oracle WSM.

Option 1: Individual Policy Definition for Each Web Service

One option is to define an individual policy for each web service and configure the request and response steps. In this option, the administrator will modify each policy for the web service and will then configure the Active Directory Authentication step during Request processing and sign the message on the Response processing.

The following screenshot shows the policy steps configured for each service.



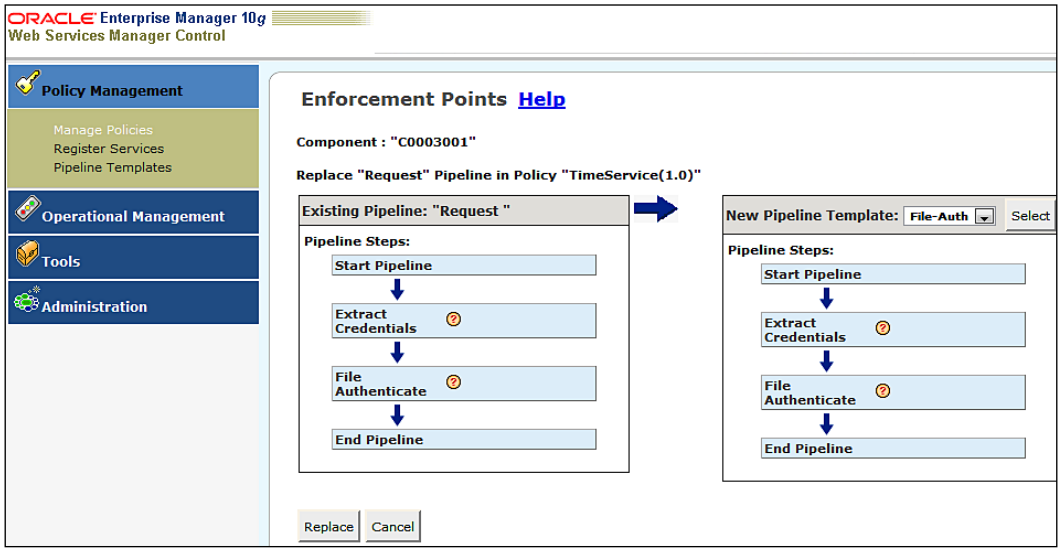
In the previous screenshot, the **Request** pipeline is configured to authenticate based on the users in the file system, and the **Response** pipeline is configured to sign the message.

In this approach, the administrator has to configure the Request pipeline for each web service and has to follow the steps correctly.

Option 2: Pipeline Templates

The other alternative to the above solution is to actually define the Request and Response pipeline template with policy steps, and even configure the policy steps. Once the pipeline templates are created, the administrator has to just modify each policy for the web service and replace the Request pipeline with the appropriate pipeline.

In this example, a Request pipeline template called "FileAuth" is created with both **Extract Credentials** and **File Authentication** steps.



The previous screenshot show the process of replacing the default **Request** pipeline for a policy with **FileAuth Request** pipeline, with steps that are already configured.

Relationship Between Policy and Service

Oracle WSM Policy Manager consists of policies to actually secure the web service. But that policies should be attached to the web service to actually enforce the security. In order to do that, the service should be registered first in Oracle WSM. Once the service is registered, a default policy is created. The default policy can then be replaced with an appropriate policy or the pipeline templates.

Oracle WSM Gateway

Oracle WSM Gateway is a self-contained module that provides protocol mediation and service end point virtualization. It can also enforce security when the client invokes the web service exposed through the gateway. The consumers of the web service can identify the web services by means of its end point URL. While a web service can be developed in Java or .NET, the URL can be virtualized by means of registering the web service within the Oracle WSM Gateway. What Oracle WSM Gateway provides is a way to virtualize the end point, thus making a highly load-balanced environment available.

Oracle WSM Gateway is no different than a traffic cop (or a security officer) from a security perspective. Oracle WSM Gateway features are:

- Perform security checks on the SOAP message
    - Inspects all the SOAP messages, as per the defined policies.
- Protocol translation
    - Transforms the HTTP request to a JMS or MQ type of request.
    - Transforms data from one format to another, if required.
- Message routing
    - Routes messages to the appropriate end point, based on the content.
    - Routing can even be dynamic, based on XPATH expressions.

The importance of Oracle WSM gateway architecture can be best explained in the context of use cases and they are:

- Proxy, or exposing internal service to external business partner, or outside of intranet
- Transport protocol translation
- Content routing
- Security policy

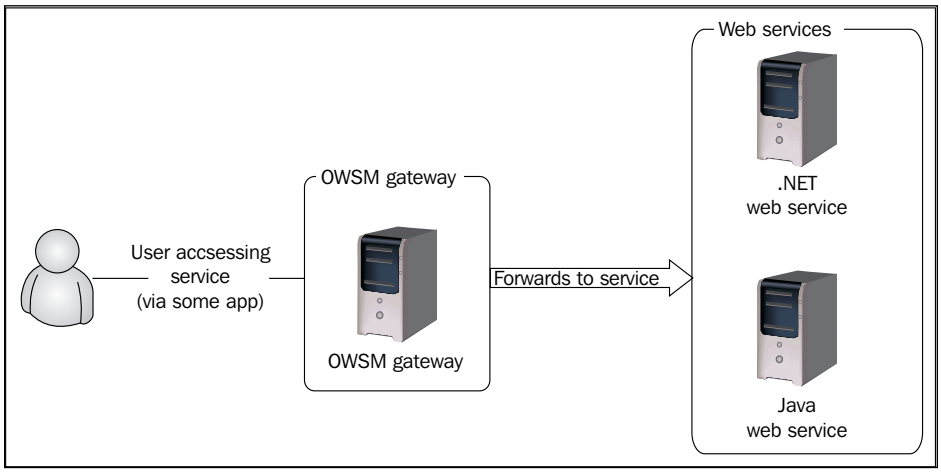
Proxy, or Exposing Internal Service to External Business Partner, or Outside of Intranet

It is not uncommon for business partners to request access to internal service and at the same time it is not uncommon for organizations to host services internally and make it available to extranet via proxy architecture. One typical example is reverse proxy architecture for web sites. Organizations can build and deploy web sites internally within their protected security zone and have a reverse proxy appliance to rewrite the URL and make it available to the external users. While a similar concept applies for web services, it is not exactly the same architecture because there is no URL rewrite in this case.

The following are the reasons why an internal service should be made available to extranet:

- Web service developed and hosted by one business unit is required by another business unit, but both of them are in different data centres without a dedicated network connection.
- Internally hosted service is a key component in enabling the process automation and needs to be exposed to the extranet.
    - Example: A consumer electronics organization typically takes orders through its own web site and has now decided to accept orders from other online portals, and it is needed to expose web services to external business partners.
- Security concerns may prevent an organization from deploying web services on its public network, and hence it is deployed internally and made available via proxy.

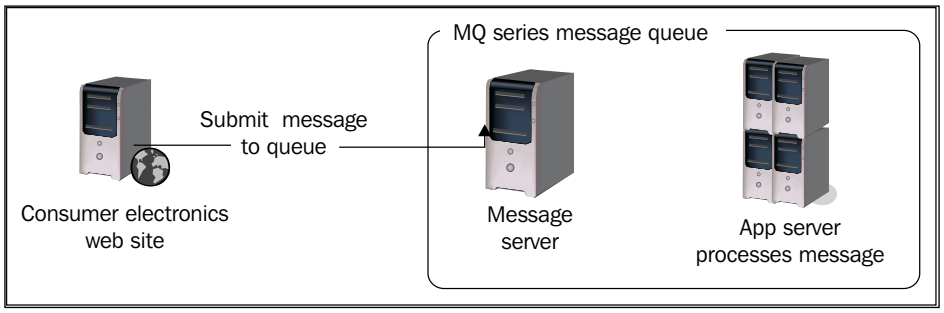
In the following figure, the Oracle WSM Gateway exposes the internal web service to an external customer. The **Oracle WSM Gateway** is accessible from the internet and the gateway then internally forwards the request to one or many internal web services.



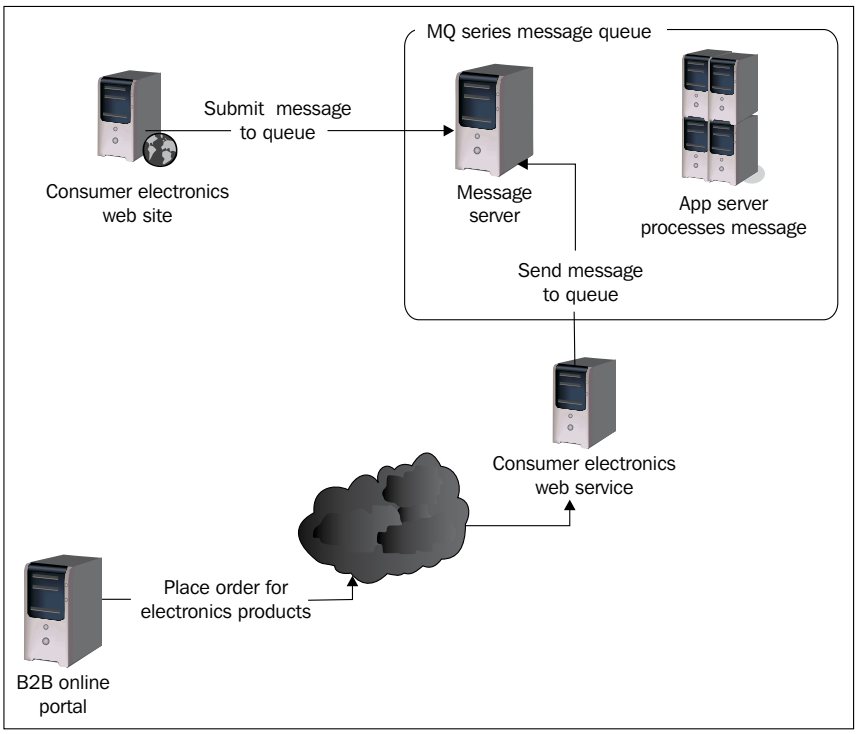
Transport Protocol Translation

The key advantage of the Service Oriented Architecture (SOA) is that we can expose the existing business process as services so that it can be better integrated with other applications and business partners. It does not mean that the existing technology should be shut down and everything should be based on web services. There are instances where internal implementation of a business process may be based on a set of internal protocols, such as MQ Series, JMS, etc. In a SOA, the architecture should create the flexibility to integrate with existing technology infrastructure, and at the same time, should adhere to the industry standards for interoperability. Let's walk through a case study to examine this in more detail.

Consider our earlier example of consumer electronics corporation which was accepting orders only from its own web site. Here the architecture was designed in such a way that once the order is placed, a message is sent to a MQ Series message queue, so that another application can process the transaction. The following figure describes the current architecture of consumer electronics corporation that leverages the MQ Series.



In this example, when the business decides to expand to receive orders from other business partners, such as online portals, it can either rewrite the application based on web services or create a web service gateway to translate the message to MQ Series so that the underlying architecture or process remains unchanged. The following figure describes how the **B2B Online Portal** can place an order via web service and the web service (gateway) then internally translates to an MQ Series request.



Protocol translation is also applicable for outgoing messages to be translated into web services protocol. This is a nice feature from Oracle WSM on its gateway architecture where you can leverage the existing applications and infrastructure, and still expose business process as services.

Content Routing

Web service end points can expose the business process as a service. However, when there are too many services exposed to business partners, it may become unmanageable from a deployment and security perspective. Consider our example of consumer electronics corporation which accepts orders from an online B2B portal. Now let's consider the following use case:

- Electronics company accepts orders for cameras, camcorders and TVs, both from its web site and from its business partners.
- Each order type is sent to a different MQ Series message queue for processing.
- In order to enable B2B Portal integration, three new web services are developed for each order type.

In the above scenario, the electronics company can either expose all the three web services to the B2B partners, or can expose just one service which then routes the message. The latter has the following advantages:

- Only one web service to be exposed to the gateway/extranet.
- Security should only be enforced on one service.
- It's easy to communicate with business partners to access one service with a different message type for each order.
- Oracle WSM Gateway can then route the message to the appropriate service internally, based on the content.

Summary

In this chapter, we discussed the Oracle WSM architecture and how services, policies, pipeline templates, policy steps and gateway are related. We also discussed the importance of gateway architecture. In the following chapters, we will explore certain individual security steps in detail, to understand how to configure the web services security.

4

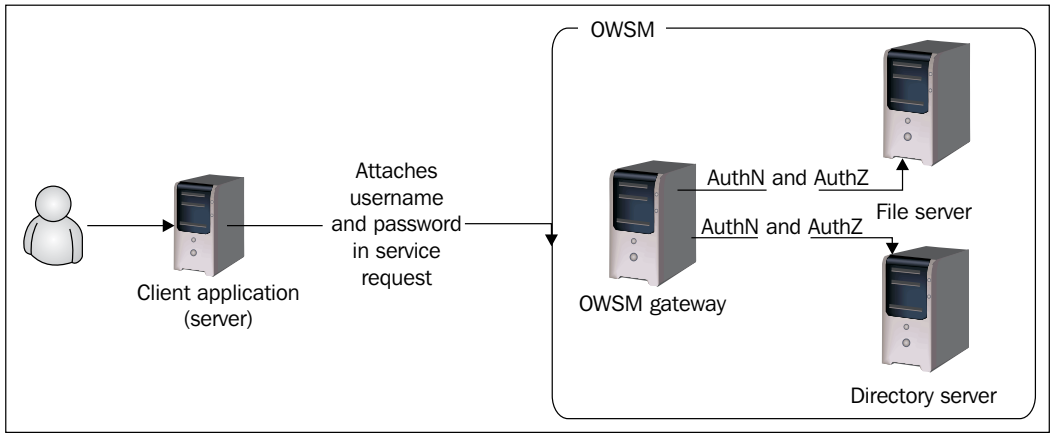
Authentication and Authorization of Web Services Using Oracle WSM

The first step in protecting web services is to authenticate and authorize the web service requests. Authentication in web services is the process of verifying that the user has valid credentials to access the web services and authorization is the process of validating that the authenticated user has appropriate privileges to access the web services. Besides restricting access to users with valid credentials and proper privileges, Oracle WSM can track who accessed which service and when—to provide detailed audit trails. In this chapter, we will explore how Oracle Web Services Manager can be leveraged to authenticate and authorize the web services requests.

Oracle WSM: Authentication and Authorization

Oracle Web Services Manager can authenticate the web services request by validating the credentials against a data store. The credentials (e.g. username and password, SAML token, certificate, etc.) that are attached to the web services will be validated against the data store, such as the file system, databases, active directory and any LDAP compliant directory. Once authentication is successful, the next step is to perform authorization by validating the username against a set of pre-defined groups which have access to the web service.

The following figure shows the process where the user accesses an application which acts as a client for the web service. The client application then attaches the username and password to make the web service request. The username and password are then validated against file system or LDAP directory by Oracle WSM, either using the gateway or the agent.



The authentication and authorization against different directory stores can be configured using Oracle WSM policy steps. Oracle Web Services Manager has predefined policy steps for:

- File Authenticate and Authorize
- Active Directory Authenticate and Authorize
- LDAP Authenticate and Authorize

In the previous figure, the **Oracle WSM Gateway** is used to protect the web services and externalize the security. In order to authenticate and authorize requests to web services, the web services can be registered within the gateway and the request pipeline of gateway will validate the credentials and authorize the access before it forwards the request to the actual web service provider. The gateway steps for authentication and authorization can be summarized as:

- Log incoming request (optional)
- Extract credentials (get the credentials from the SOAP message or HTTP header)
- Authenticate (file authenticate, active directory authenticate, etc.)
- Authorize (file authorize, active directory authorize, etc.)
- Request is forwarded to the web service provider

The response from the web service also follows through a similar response pipeline where you can implement the log, encryption of response, or signing, or response, etc. While it is not required to implement any steps in the response pipeline, there should be a response pipeline even if it's doing nothing.

Oracle WSM: File Authenticate and Authorize

Oracle Web Services Manager can authenticate the web services requests against a file that has the list of usernames and passwords. In this example, the username and password information are part of the SOAP message, however one can also send a username and password as HTTP header, or it can be any XML data that is a part of the web services message. While file-based authentication can easily be compromised, it is often used as a jump start or testing process to validate the authentication and authorization process.

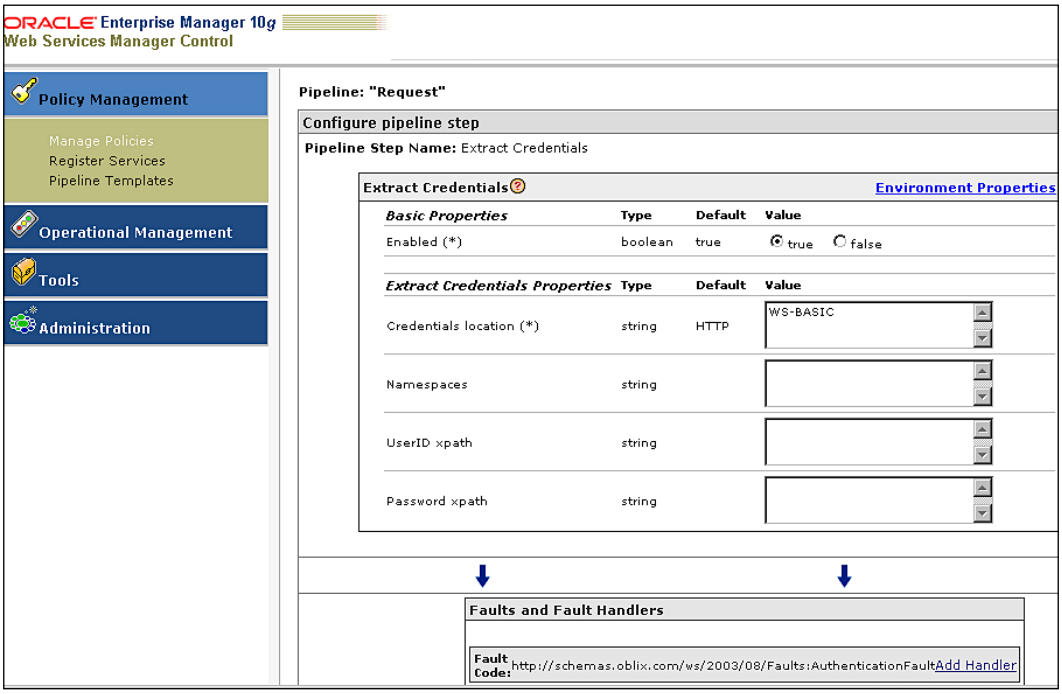
Authentication and authorization of web service requests against a file requires three main steps, and these are described below. There is a default log step which will log all the request and response messages, and you can also include that log step at any point to log messages:

- Extract Credentials
- File Authenticate
- File Authorize

The first step to authenticate a web service request against a password file (file authenticate) is to extract the username and password credentials from the SOAP message. The client application attaches the username and password to the SOAP message, as per the UserName token profile, which is described later in this chapter.

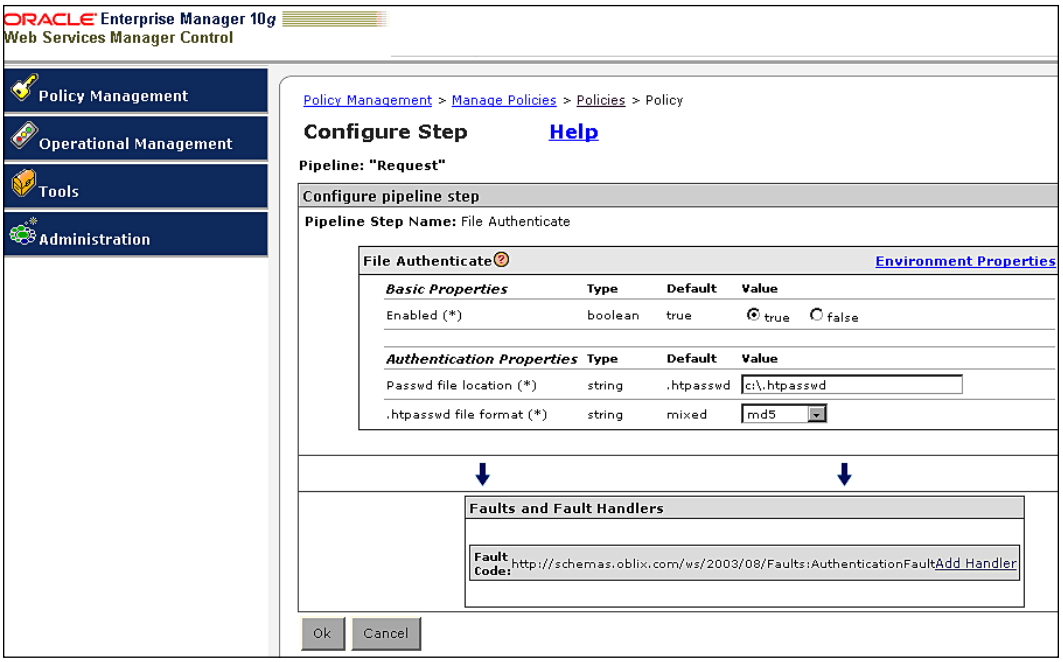
In the policy to authenticate the web service against the file, add the step in the request process to extract credentials. Since this is a web service request, as opposed to HTTP post, configure the **Credentials location** to **WS-BASIC** (refer to the following screenshot).

Note: WS-BASIC means that it is WS-security compliant. WS-security is the oasis specification that specifies how security tokens are inserted as a part of the SOAP message. In other words, WS-BASIC means that the username and password can be found in the SOAP message, as per the username token profile of the WS-security specification.



Once the credentials are extracted, the next step is to validate them against the file. The default implementation of the Oracle WSM File Authenticate requires the username and password to be in a comma separated format and the password should be the hash value using a MD5 or SHA1 algorithm.

In order to authenticate the credentials against the data store, the next step is to configure the **File Authenticate** step in Oracle WSM. In this step, the options are straightforward. We have to configure the location of the password file and the hash algorithm format as either **md5** or **SHA1** (see the next screenshot).



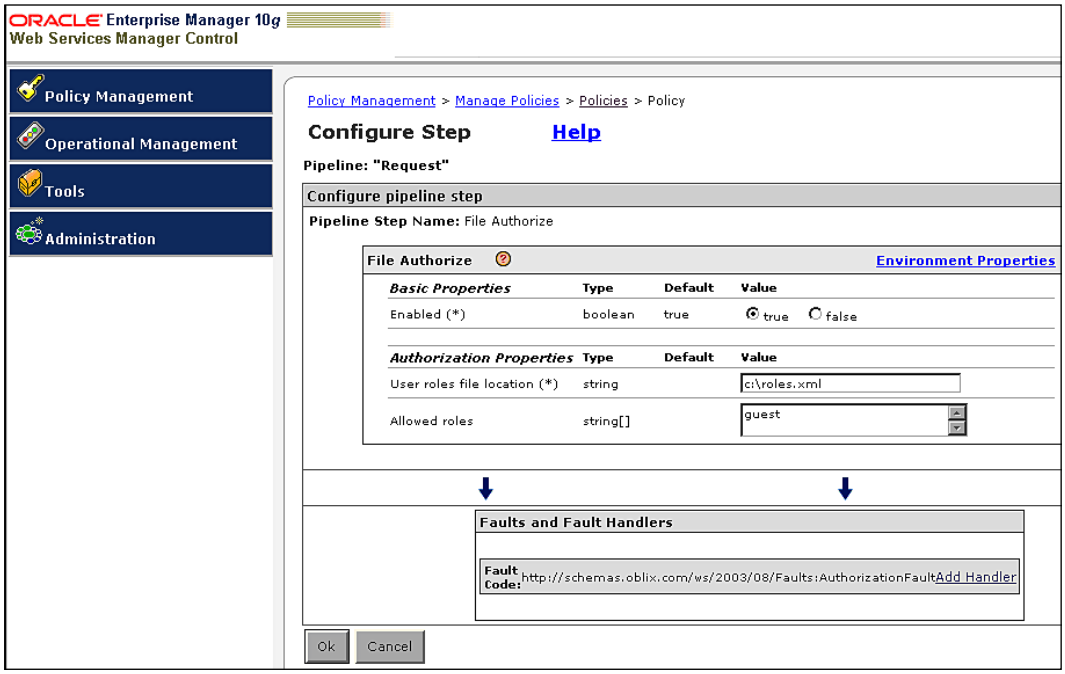
The sample file with username and password is:

```
bob:{MD5}jK2x5HPF1b3NIjcmjdlDNA==
```

Note: You can use the wsmadmin tool provided as part of Oracle WSM (standalone or SOA suite). Type: `wsmadmin md5encode bob password c;\.htpasswd`

Now that the authentication steps are configured, the next step is to configure the authorization policy step to ensure that only valid users can access the web service. For the file authorization method, it is no different than the file authenticate method i.e. even the user-to-role mappings are kept in the file.

The following figure shows the **File Authorize** policy step. In this step, we have to define the location of the XML file that contains the users to roles mapping, and also the list of roles that should be allowed to access the service.



The roles XML file should look like:

```
<?xml version='1.0' encoding='utf-8'?>
<UserRoles>
  <user username="joe" roles="guest"/>
  <user username="Bob" roles="Admin,guest"/>
</UserRoles>
```

In the previous XML file, the list of roles the user belongs to are defined as a value of **`roles element`** and is **`comma`** separated.

Now that we have completed the steps to extract credentials, authenticate the request and also authorize the request, the next step is to save the policy steps and commit the policy changes. Once the policy is committed, any request to that web service would require a username and password, and that user should have necessary privileges to access the service.

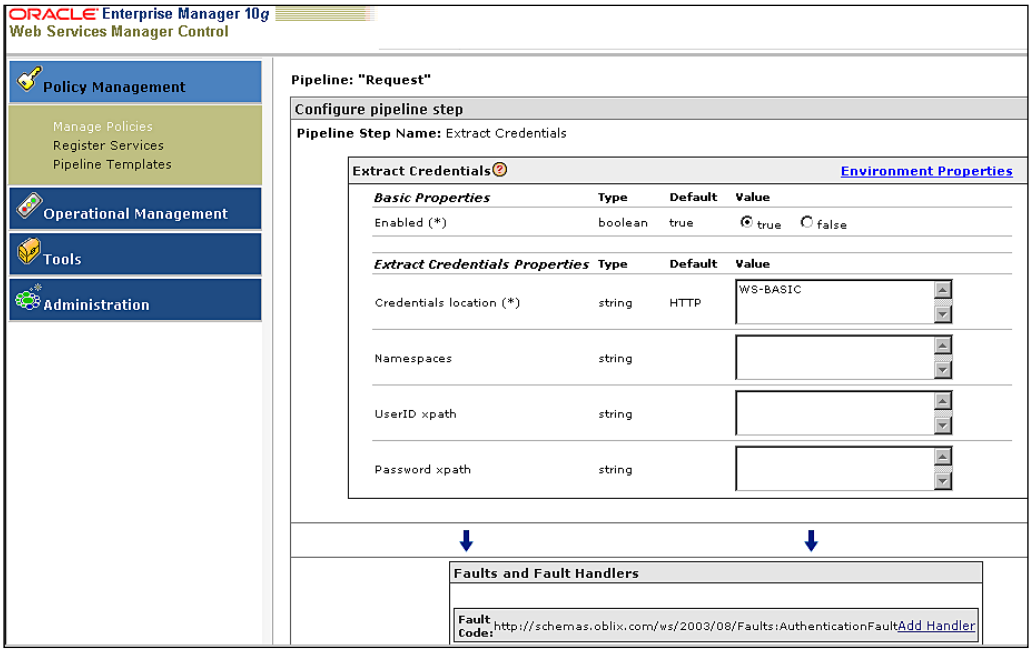
Oracle WSM: Active Directory Authenticate and Authorize

In the previous section, we discussed authenticating and authorizing web service requests against a file. Though it's an easy start, security based on a file system can be easily compromised and will be tough to maintain. Authentication and authorization of web services are better handled when integrated with a native LDAP directory, such as active directory, so that the AD administrator can manage users and group membership. In this section, we will discuss how to authenticate and authorize web service requests against an active directory.

Active-directory-based authentication and authorization of web service requests involves the same steps as file-based-authentication and authorization, and they are:

- Extract Credentials
- Active Directory Authenticate
- Active Directory Authorize

The first step towards active-directory-based authentication and authorization is to extract the username and password credentials from the SOAP message. This step is the same for both file-based-authentication and active-directory-based authentication. The next screenshot describes how the **Extract Credentials** steps should be configured.

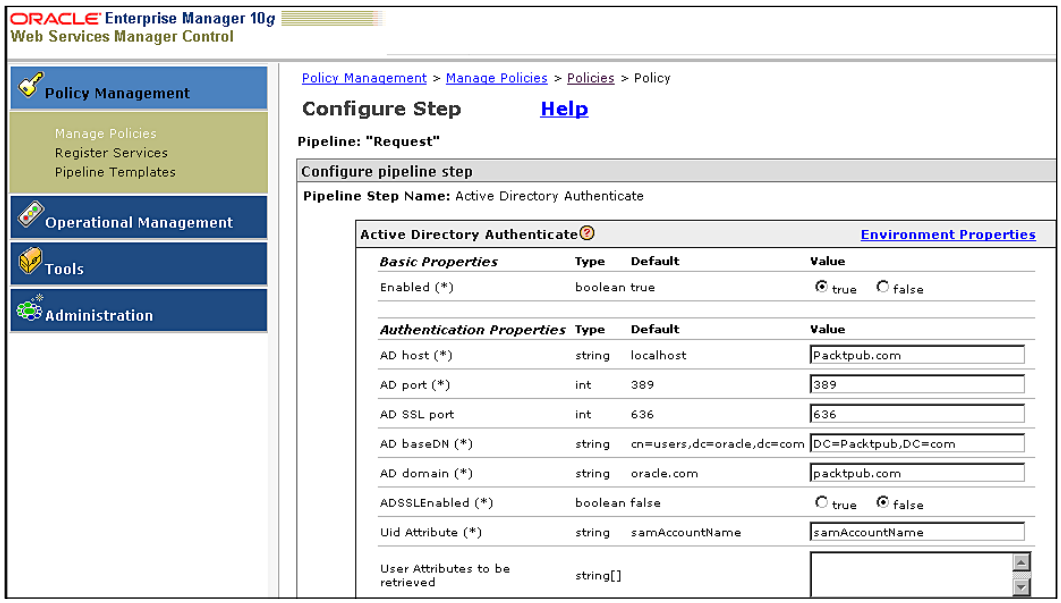


In the **Extract Credentials** step, the **Credentials location** should be changed to **WS-BASIC** since it is a web service invocation.

The second step is to authenticate the web service request against the active directory data store. Well, we know that the credentials i.e. username and password are attached to the SOAP message. Once the credentials are extracted from the SOAP message, we then have to define the following in order to authenticate the user against the active directory:

- Name of the active directory server
- Ports on which to communicate (both plain text and SSL)
- Base distinguished name under which the user can be found
- Active directory domain name
- User attribute that uniquely identifies the user.

This information is then configured in the **Active Directory Authenticate** policy step.



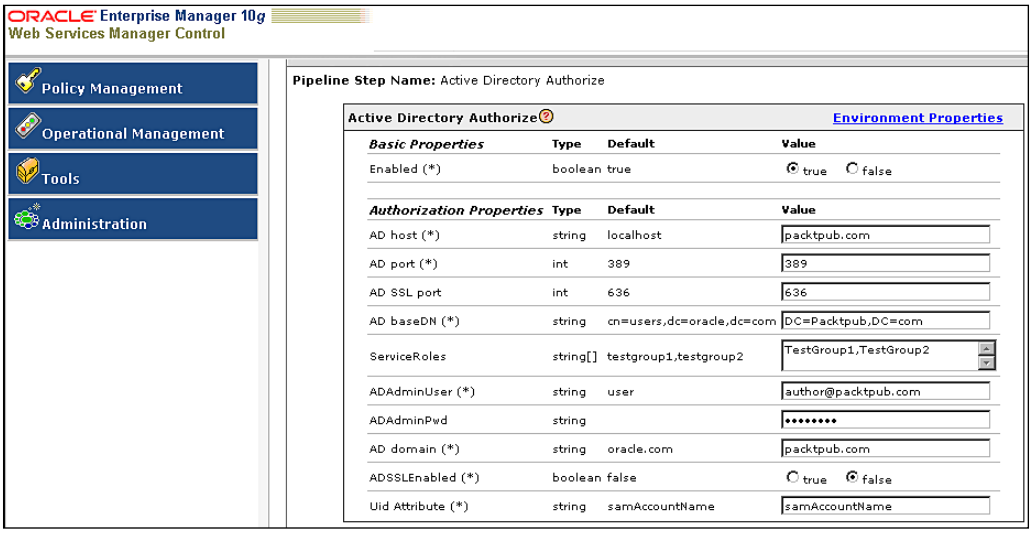
In the previous figure, the sample **Active Directory Authenticate** policy step is configured to validate the user against:

- Packtpub active directory at **Packtpub.com**
- Listening on port **389** and **636**
- With the domain name of **packtpub.com**

- The user can be found under **DC=Packtpub, DC=com**
- The unique user id attribute is **samAccountName**

Once this policy step is configured, the username and password will be validated against the packtpub active directory.

The next step is to ensure that only valid users can access the web service by defining the Active Directory Authorize policy step. In this step, Oracle WSM will validate the authenticated user against the configured groups and will be granted access only if the user belongs to one of those groups. The following screen capture shows how the **Active Directory Authorize** step is configured.



In the previous figure, the AD Authorize policy step is configured to protect against **TestGroup1** and **TestGroup2**. In the This step, you should also define the service account that can bind to the active directory.

Note: The **AD baseDN** should be a root distinguished name under which both users and groups are defined. Technically, users can be in a different container to the groups, but in the AD Authorize policy step, the **AD baseDN** should be the root of both the user's and group's container.

Once the AD Authenticate and Authorize steps are configured, the next step is to save the policy steps and then commit the policy. When the web service request is made with the appropriate username and password that belong to TestGroup1 or TestGroup2, they will be either granted or denied access to the service.

In the next section, we will take a closer look at what policy templates are and how they can be used in defining policy steps.

Oracle WSM: Policy Template

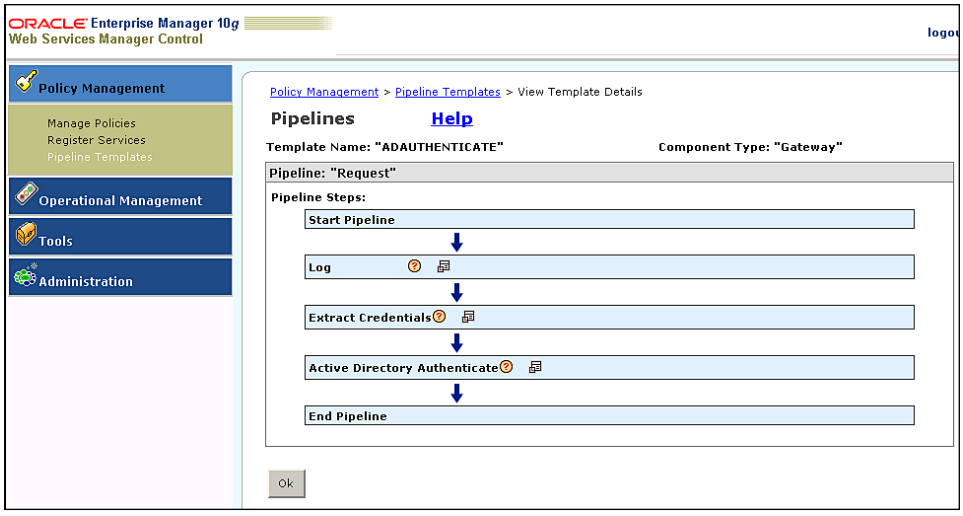
Policy templates are reusable policy steps that can be created once and used across multiple policies. It is not uncommon to have a similar sequence of steps to perform operations such as authentication, authorization, signature verification, etc. Instead of defining the same steps over and over again in each policy, it can be defined as a policy template in one place and then used when defining each policy step.

In the previous sections, we discussed authenticating users against an active directory. Imagine that each service should be authenticated against the same active directory; one has to define the same steps of Extract Credentials and AD authenticate across each policy. While Extract Credentials and AD Authenticate are mandatory, there is also an optional logging step that may be required to help troubleshoot the application. In the overall, the administrator has to:

- Create the log step and specify what information should be logged.
- Create the Extract Credentials steps in each policy and define if it is WS-BASIC or not.
- Create the AD Authenticate step and define the same AD user DN, uid attribute, etc.

When these steps are repeated across all services, it increases the administrative overhead and can also cause increase in the chances of manual data entry errors.

Defining the policy template for AD Authenticate for your environment would be ideal so that the default request template can be replaced with this new template. In order to define the policy template, the administrator has to go to the **Pipeline Templates** section and then create a new template.



In the previous figure, a new request pipeline template named **ADAUTHENTICATE** has been defined. Defining the pipeline template is exactly the same as configuring the steps in policy in the request pipeline section.

In this example, we defined a template that will:

- **Log** the request.
- **Extract Credentials** with WS-BASIC.
- Authenticate the user against the active directory using samaccountname as uid attribute.

Once this template is defined, in the policy section, when you want to authenticate the request against the active directory, you can replace the default request pipeline template with the ADAUTHENTICATE pipeline template.

In the previous figure, the default **Request** pipeline is replaced with the ADAUTHENTICATE pipeline template.

Note: Once the template is chosen, it can then be modified to add any additional steps, and that will not affect the actual template.

Oracle WSM: Sample Application AD Authentication

The previous sections described the various options and steps involved in authenticating web service requests. In this section, we will create a sample Microsoft .NET client application that will access a web service whose security is managed by Oracle Web Service Manager for authenticating the requests.

Since this book assumes that the readers are familiar with the creation of web services, we will leave out the details of web service creation. We will be using the sample web service that comes with Oracle WSM product installation.

We will divide this section into two components; one is to ensure that the web service is protected by Oracle WSM and the other component is the creation of the .NET client application.

Web Service Security Policy

In order to secure the web service, it should be defined within Oracle WSM and then the policy will be attached and modified to protect the web service. Three high level steps are required to protect a web service using Oracle WSM, and they are:

- Register the web services with Oracle WSM.
- Create and define the security policy.
- Commit the policy.

Registering The Web Service with Oracle WSM

The first step in securing the web service is to register the web service with Oracle WSM as a part of a gateway. In order to register the web service, one has to have the service `WSDL` URL handy. In our sample, we will be using the time service that comes with the Oracle WSM installation. You can download and install Oracle Web Service Manager from `http://download.oracle.com/otn/nt/ias/101310/soa_windows_x86_ws_mgr_101310.zip` Once the Oracle WSM is installed, you can start the Oracle WSM from the `bin` directory of the installation folder by entering

`wsmadmin.bat start`.

Once the Oracle WSM is started, you can access it via browser by typing

`http://localhost:3115/ccore` The default username and password is admin/oracle.

You can now view the Time Service at:

```
http://owsm.packtpub.com:3115/ccore/TimeService.wsdl
```

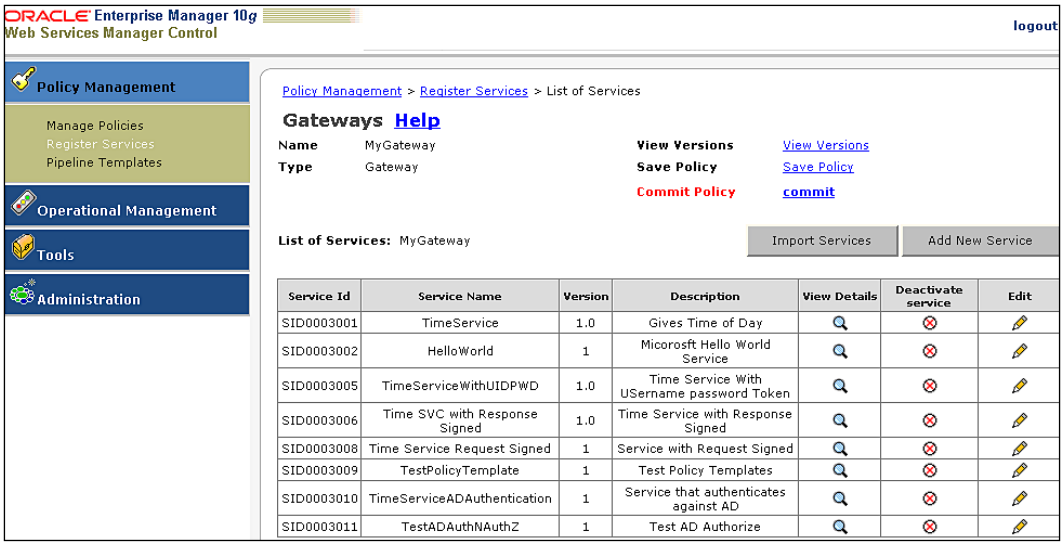
The following steps describe how to register the service.

Login to Oracle Web Service Manager Console at:

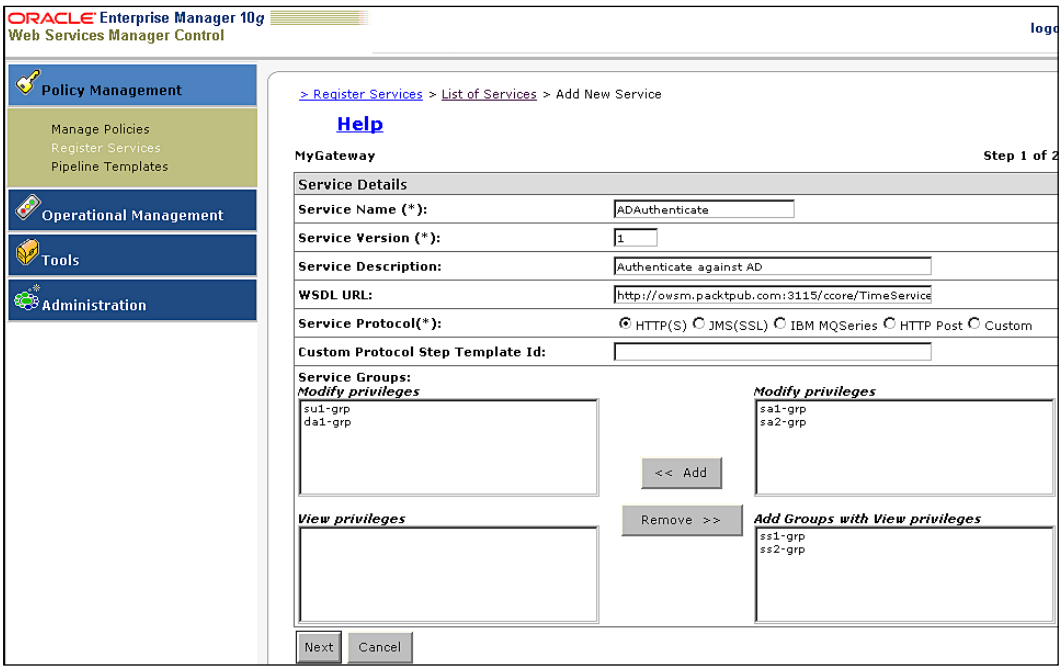
```
http://owsm.packtpub.com:3115/ccore
```

Click on **Policy Management** and then **Register Services**, you will see the list of gateways that are available, as shown in the next screen capture.

On the right side of the screen, if you click on the **Services** hyperlink, you will see the list of registered services.



In order to add a new service, click on **Add New Service** from the right side panel (refer to the previous screenshot).



The previous screen shows the details that can be added while adding the new service. The * after each label makes those fields mandatory. The screen asks for typical information such as:

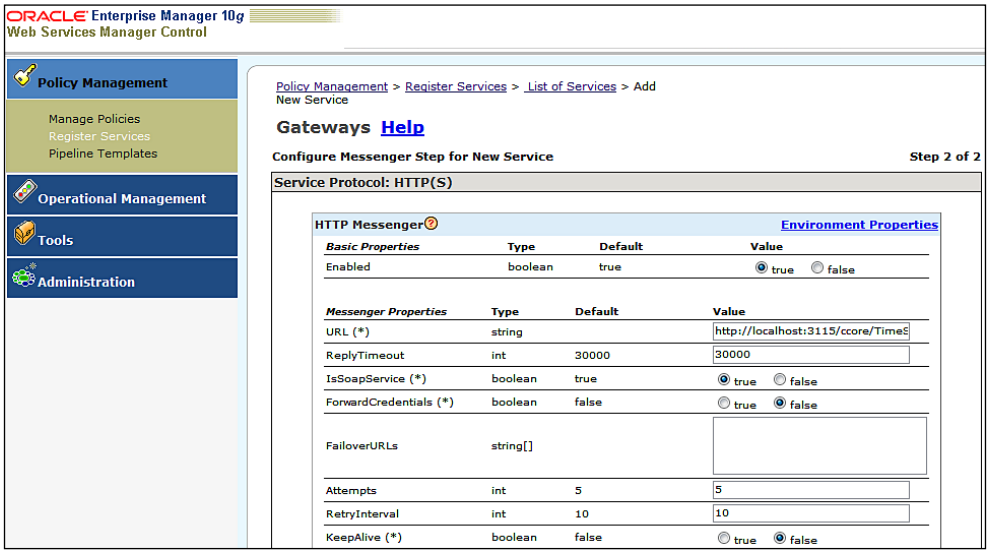
- Name of the service
- Version of the service
- Any description of the service
- **WSDL URL** of the service
- Protocol in which the service will accept messages

It also asks for additional information, such as **Service Groups**, groups that are part of Oracle WSM who have the right to view and who have the right to update.

In our example, we are registering a time service that will authenticate users against the active directory. The time service is registered with the following information:

- Name of service: ADAuthenticate
- Service version: 1
- Service description: Time service that will authenticate users against AD
- WSDL URL: `http://owsm.packtpub.com:3115/ccore/TimeService.wsdl`
- Service protocol: HTTP(S)
- Service groups: Select the default that has full permissions

Once the information is filled out, click **Next** on **New Service** registration. This will take you to the next screen which will display the actual **URL** of the service (refer to the next screenshot).

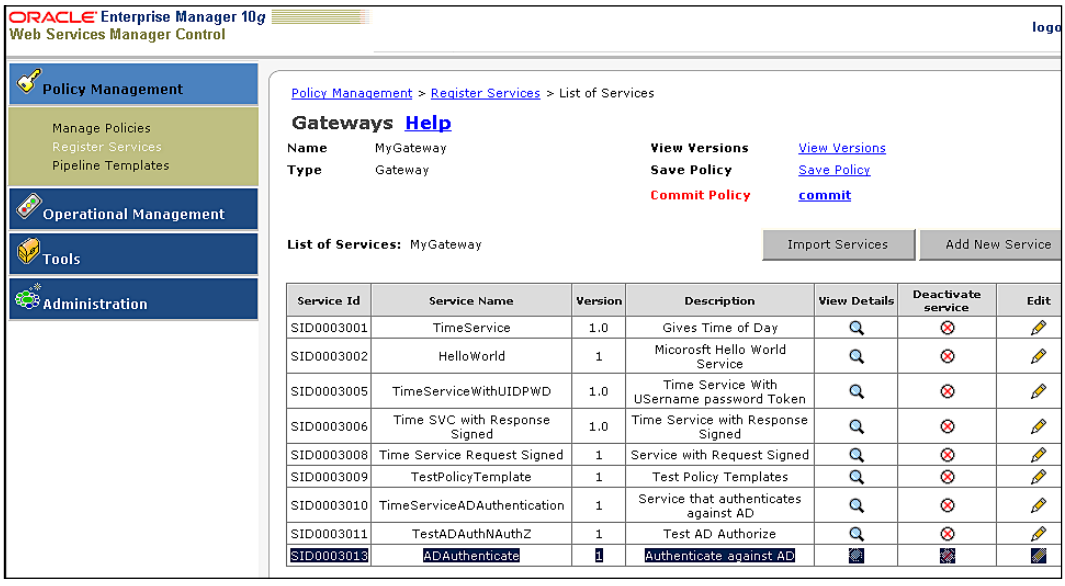


The URL in this page comes from the WSDL URL. You just have to make sure that the service is enabled and find out whether it is SOAP service or not.

You can then click **Finish** to register the service. Once you click **Finish**, the Oracle WSM internally generates a new service ID, and now the client applications can use that service ID to communicate (refer to the following screenshot).

The previous figure shows that Oracle WSM registered the time service and created a new service ID as **SID00003013**.

Click **OK** to get back to the main screen that lists all the services.

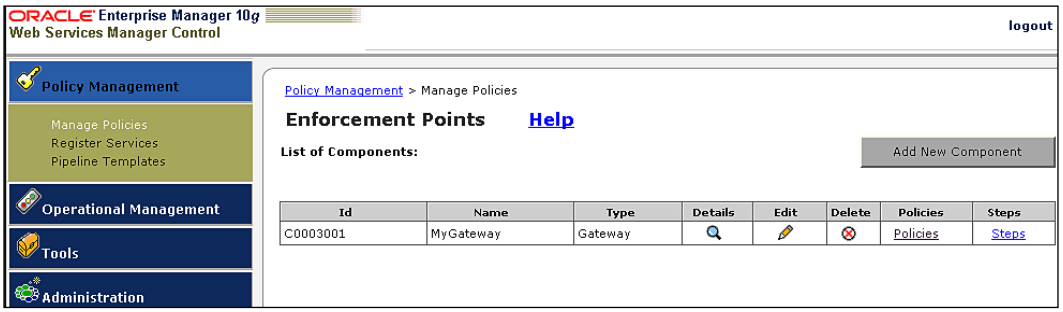


The previous figure shows all the services. The highlighted one is the **ADAuthenticate** service that was just registered.

We have only added the new service which hasn't been committed yet. The previous figure shows the **Commit Policy** in red and you can now click on that to commit the policy.

Creating The Security Policy

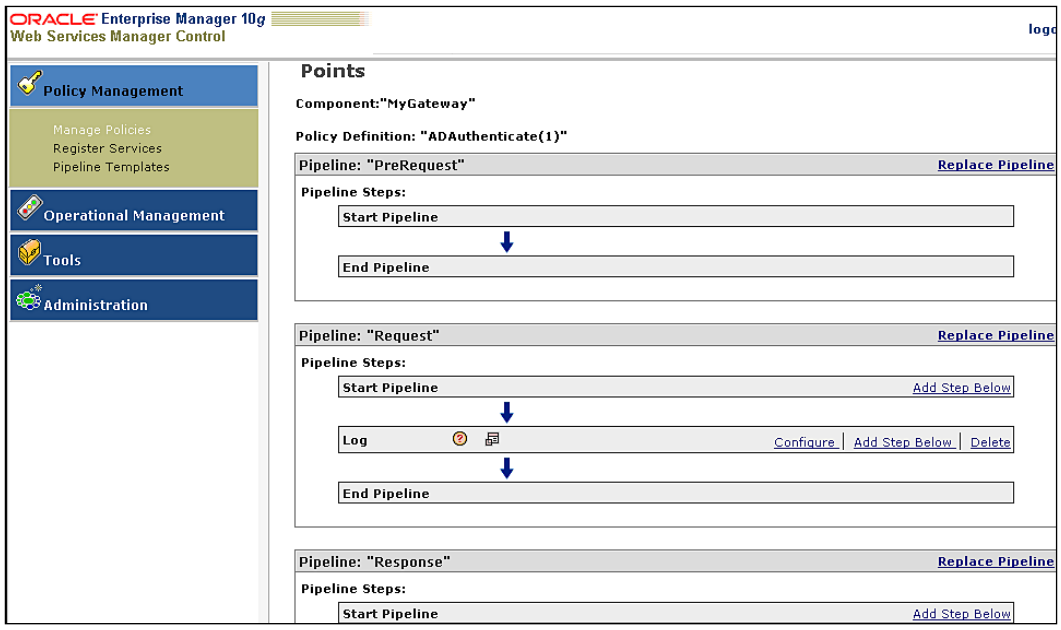
In the previous section, we learnt how to register the service and commit the policy. Once a service is registered and committed, it is associated with a default policy without any security. You can actually view the associated policy from **Policy Management**. In order to view the policy, you can click on **Policy Management** and then **Manage Policies**. This will bring you to the screen with the gateway information and a hyperlink for policies.



You can then click on **Policies** to view all the policies and you will see the **ADAuthenticate** policy that is created by default (refer to the following screenshot).

At this point, a dummy policy is attached to the newly registered ADAuthenticate service without any security. Any application that can call this web service will not be forced to provide the username and password.

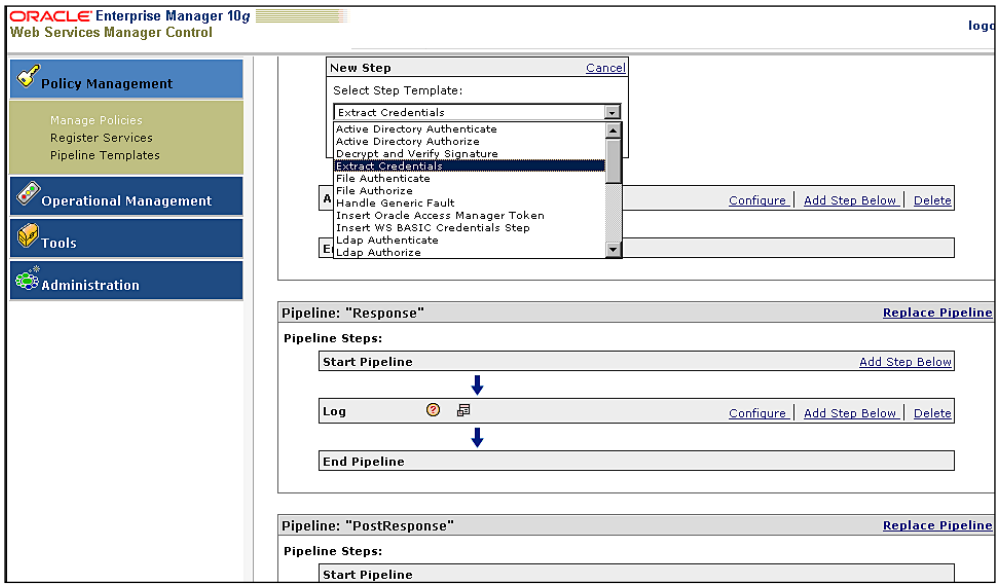
In order to enforce the authentication policy, you have to edit it to make sure that Oracle WSM checks whether the SOAP message request has a valid username and password. You can now click on the **Edit** button to edit the policy, and a new screen is shown (refer to the following screen capture).



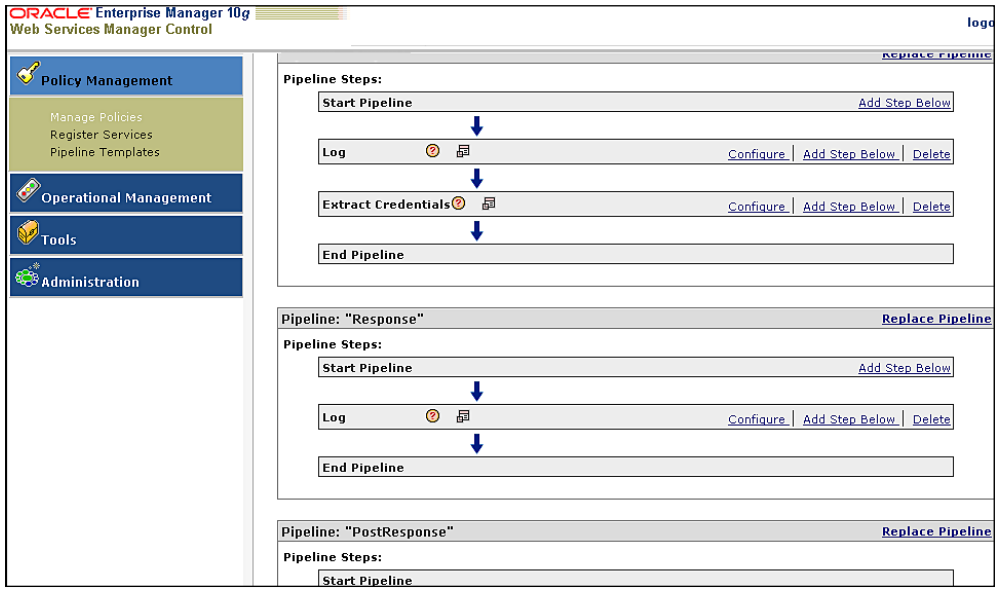
The Oracle WSM web service policy goes through four pipelines where the message can be manipulated or security can be enforced. In the previous figure, you can see the **Pipelines** for **PreRequest**, **Request**, **Response** and **PostResponse**. Typically it's the **Request** and **Response** pipeline that is often used to either validate the incoming message (authentication, authorization, encryption, and signature) or perform security on the outgoing message. Note: **PreRequest** and **PostReponse** will be disabled in future releases.

In our case, we need to validate the username and password in the incoming message against the active directory before the client application is given access to the service. In order to authenticate the request, we need to configure the request pipeline and add the **Extract Credentials** step and **AD Authenticate** step.

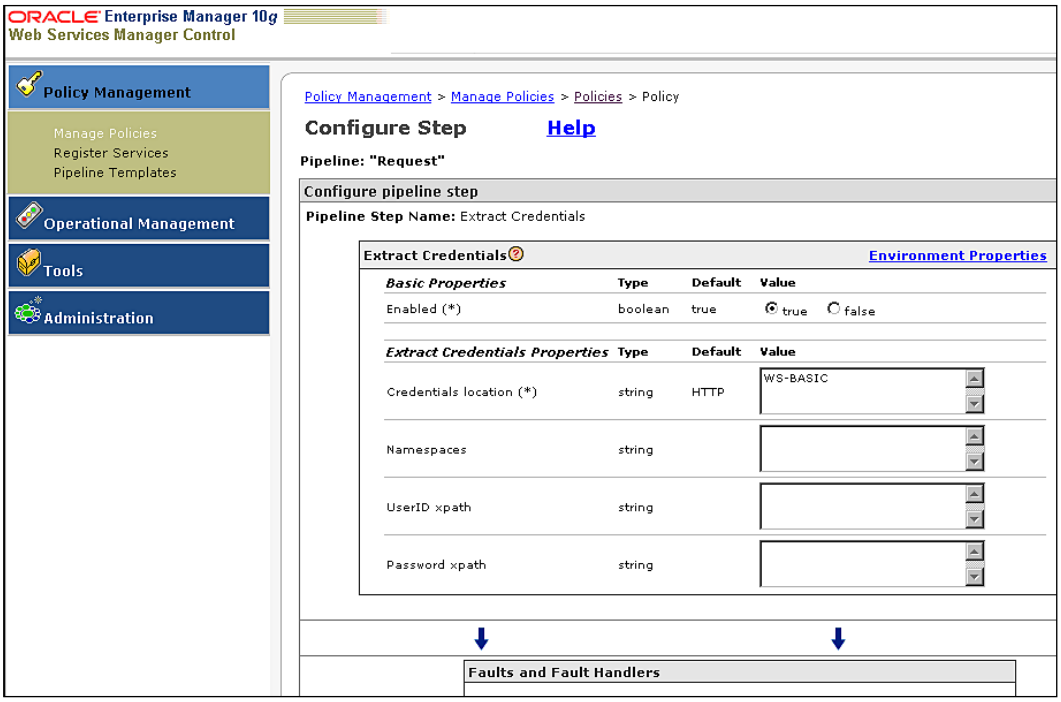
We can add the **Extract Credentials** step by clicking on the **Add Step Below** hyperlink under the **Log** step. Once you click the **Add Step Below**, you can then select the **Extract Credentials** from the dropdown and click **OK** (refer to the following screenshot).



Once you click **OK**, the **Extract Credentials** step is added below the **Log** step in the **Request** pipeline, and it will look as shown in the following screenshot.

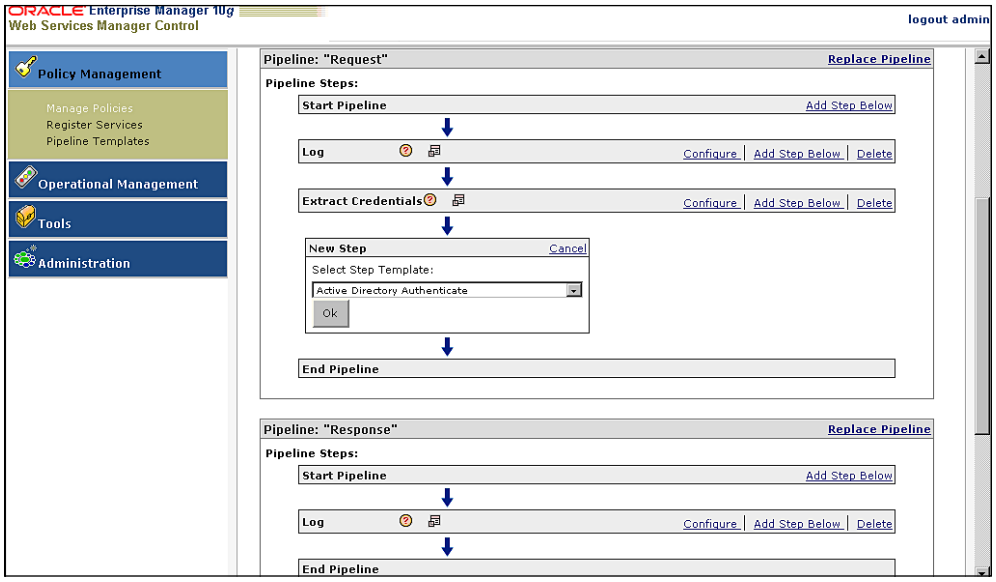


Once the **Extract Credentials** step is added, you can configure the step to modify the default values by clicking on the **Configure** hyperlink. Once you click on the **Configure** hyperlink, you will see the default values which can be replaced as shown in the following screenshot.

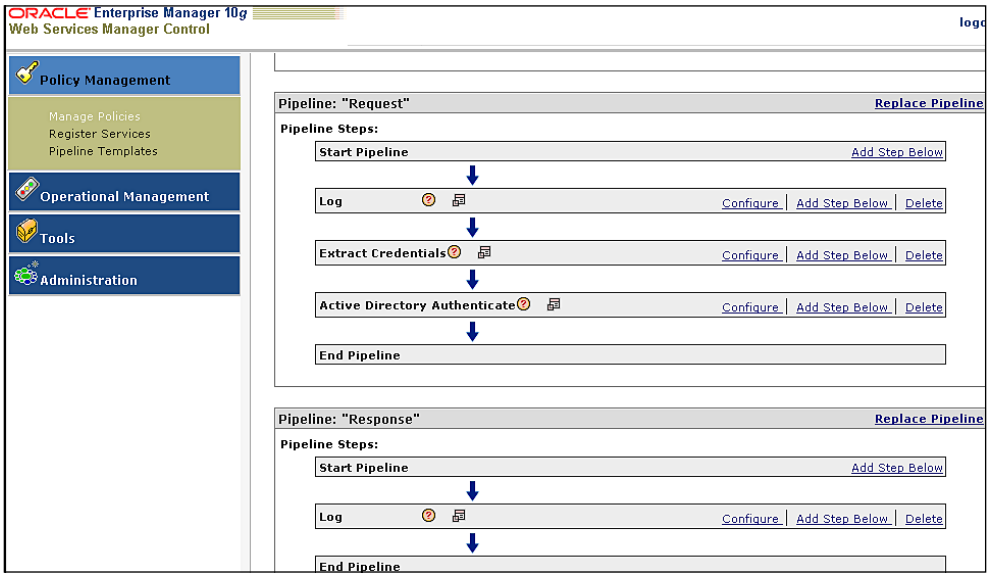


In the previous figure, the **Credentials location** is now changed to **WS-BASIC**. Since we set the service as "SOAP service" during the registration, the username and password that will be sent can be obtained from the SOAP message. You can now click **OK**; you can add the next step.

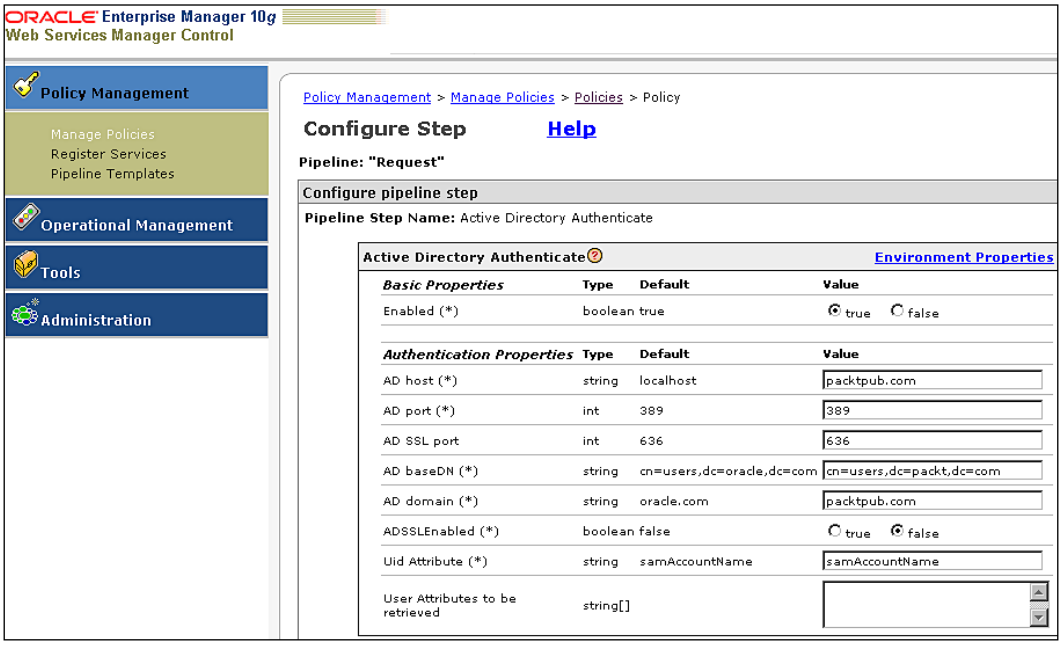
Now that the credentials are extracted, we need to validate them by adding the **AD Authenticate** step. You can now click on the **Add Step Below** under **Extract Credentials** and select **AD Authenticate** from the dropdown as shown in the following screen capture.



Once you click **OK**, **AD Authenticate** is added below the **Extract Credentials** (refer to the following screenshot).



Now we have to modify the default values with the actual active directory server name, domain name, username attribute in active directory, the **OU** under which the user can be found.



In the previous figure, the fields that are mandatory are marked with **(*)** and we have the following values populated as:

- **AD host**: active directory server name: **packtpub.com**
- **AD port**: active directory port: **389**
- **AD SSL port**: secure port where the communication happens over secure channel: **636**
- **AD baseDN**: DN under which users can be found: **cn=users**, **dc=packtpub**, **dc=com**
- **AD domain**: domain name of AD: **packtpub.com**
- **ADSSLEnabled**: **true** or **false** to specify if AD can support secure communication: **false**
- **Uid Attribute**: unique attribute that uniquely identifies the user: **samAccountName**
- **User Attributes to be retrieved**: any additional user data to be retrieved: **none**

Now you can click OK and finish updating the policy.

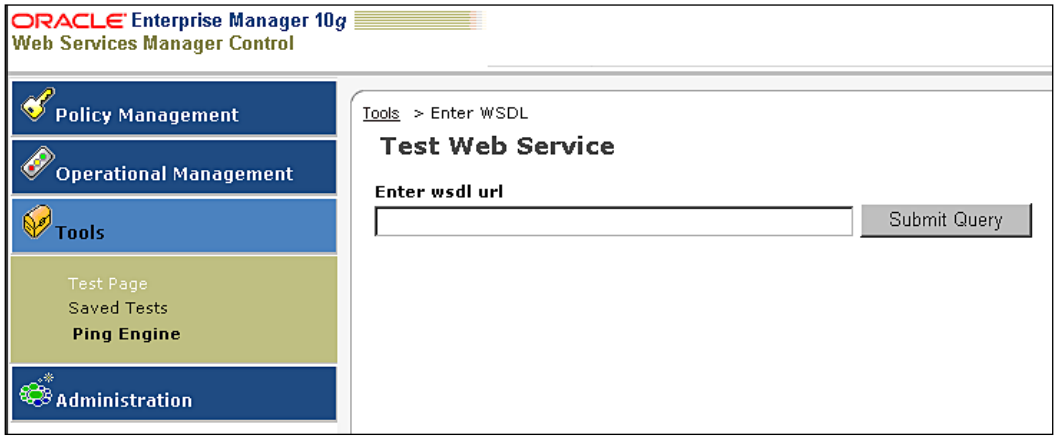
Commit The Policy

You still have to commit the policy before it can take effect. Once the policy is committed, all the requests to time service will require a valid username and password. In the next section, we will describe how to use Microsoft .NET Windows Forms as a client application to access the web service with the username and password attached to the web service request.

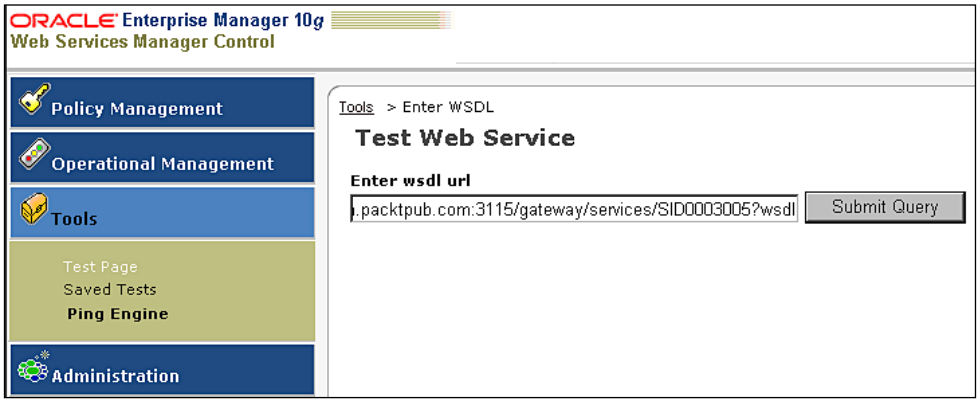
Oracle WSM Test Page as Client Application

In order to test the policy that was just defined (file based authentication or active directory authentication), one has to write a client application that will invoke the web service with appropriate credentials. While writing client applications is certainly possible, Oracle WSM ships with its own test page where you can perform certain basic tests. In this section, we will explore how to use the test page to test the web services that validate the user's credentials (authentication) and grant access to valid users (authorization).

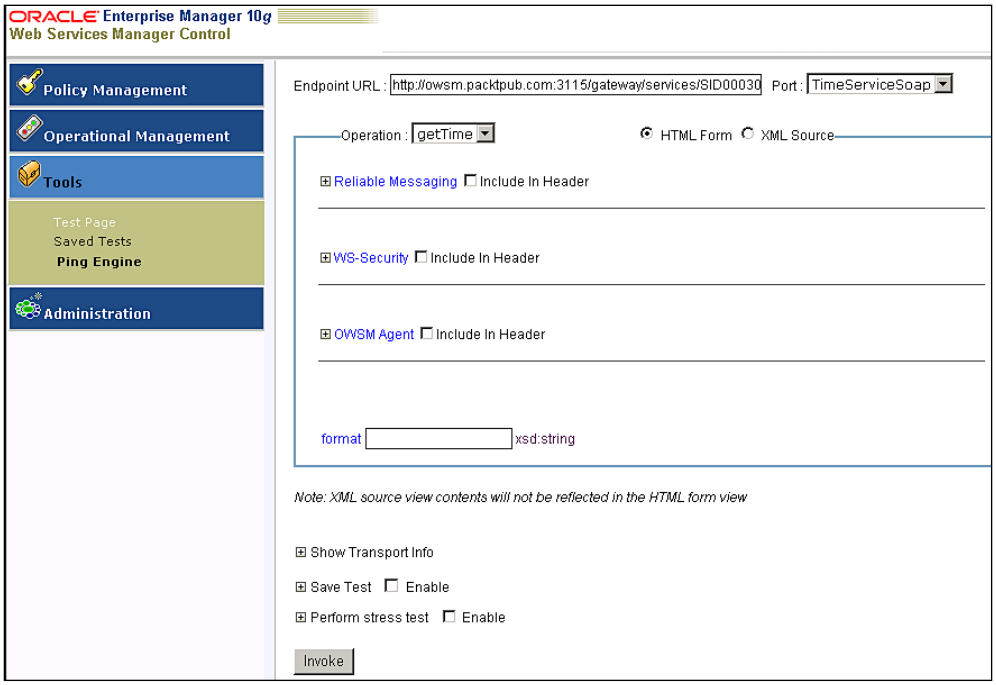
The **Tools** menu from the Oracle WSM has the **Test Page** link. The following figure shows what the test page looks like:



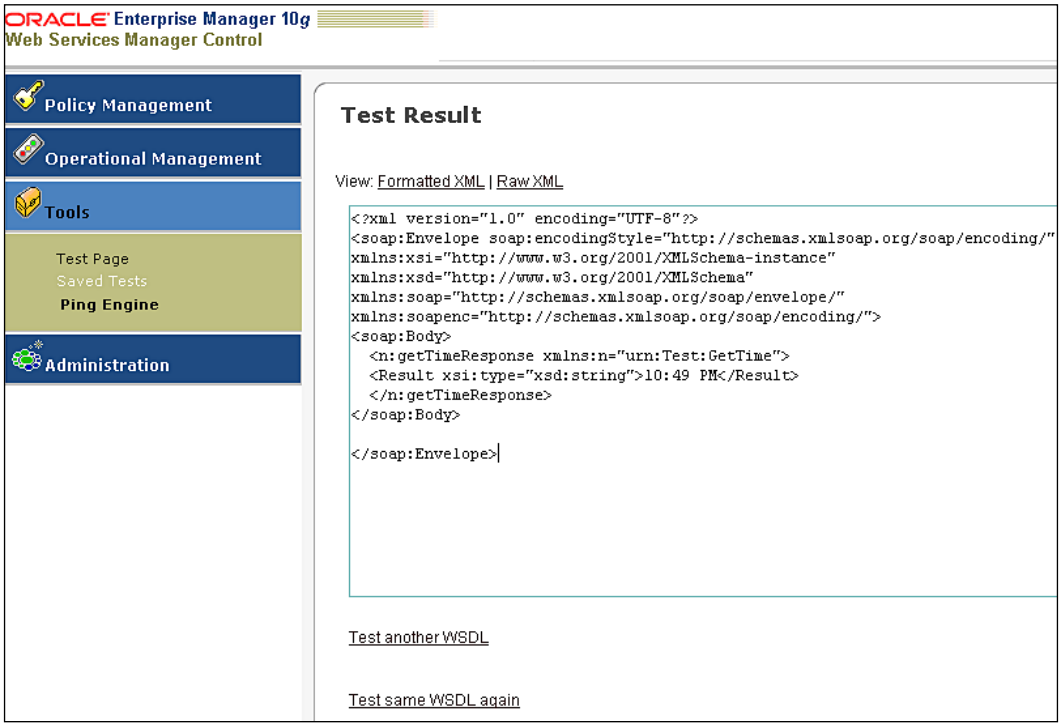
You can enter the WSDL (Web Services Description Language) here and click **Submit Query**. This will take you to the next screen where you can enter the username and password.



The WSDL entered in the previous screenshot is the WSDL for the time service where the policy requires the client to provide appropriate credentials (username and password). It will validate if the user has the privilege to access the web service. Once you click on **Submit Query**, the next screen will appear as shown in the following screenshot.



You can enter the **User Name** and **Password** under **WS-Security**. Remember that we chose WS-BASIC while configuring the policy, which will send the username and password as per the WS-security specification. There is also an option to save the test and **Perform stress test**, etc. When you click **Invoke**, the web service will be invoked, the credentials will be validated, and the response message will be displayed as shown in the following screen capture.

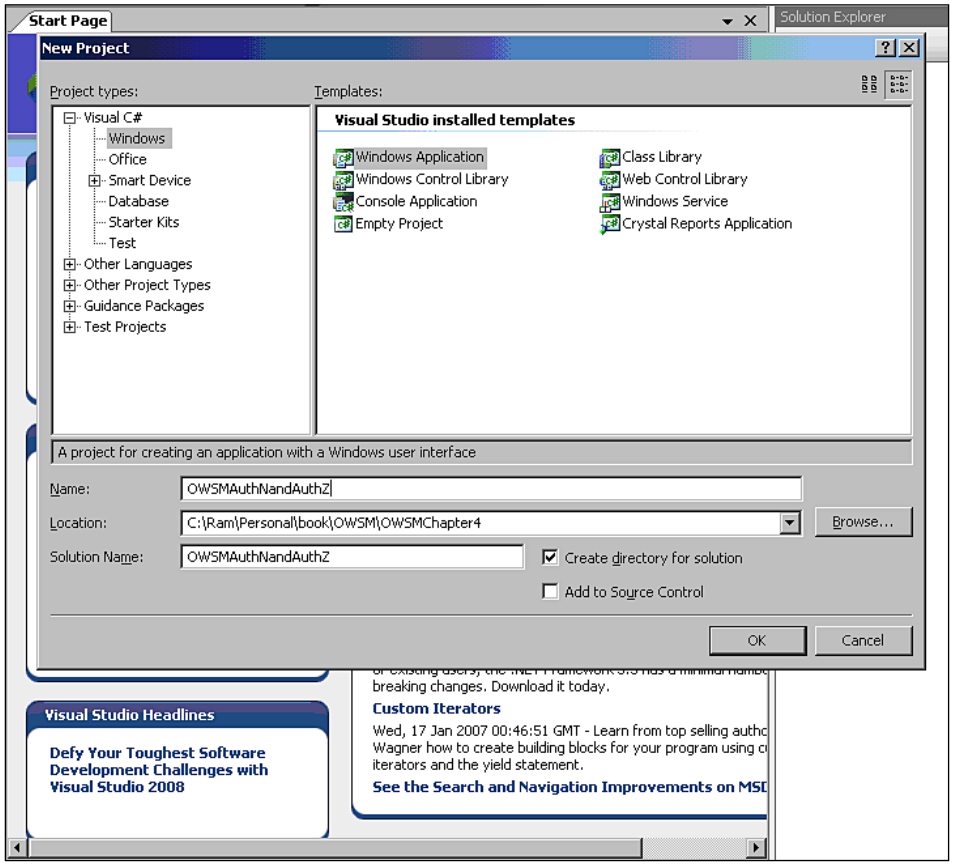


While the Oracle WSM test page is an excellent tool to test web service policies, you can also write a client application to test the web services. In the next section we will describe how to use Microsoft .NET to create a client application which will invoke the web service with a username and password.

Microsoft .NET Client Application

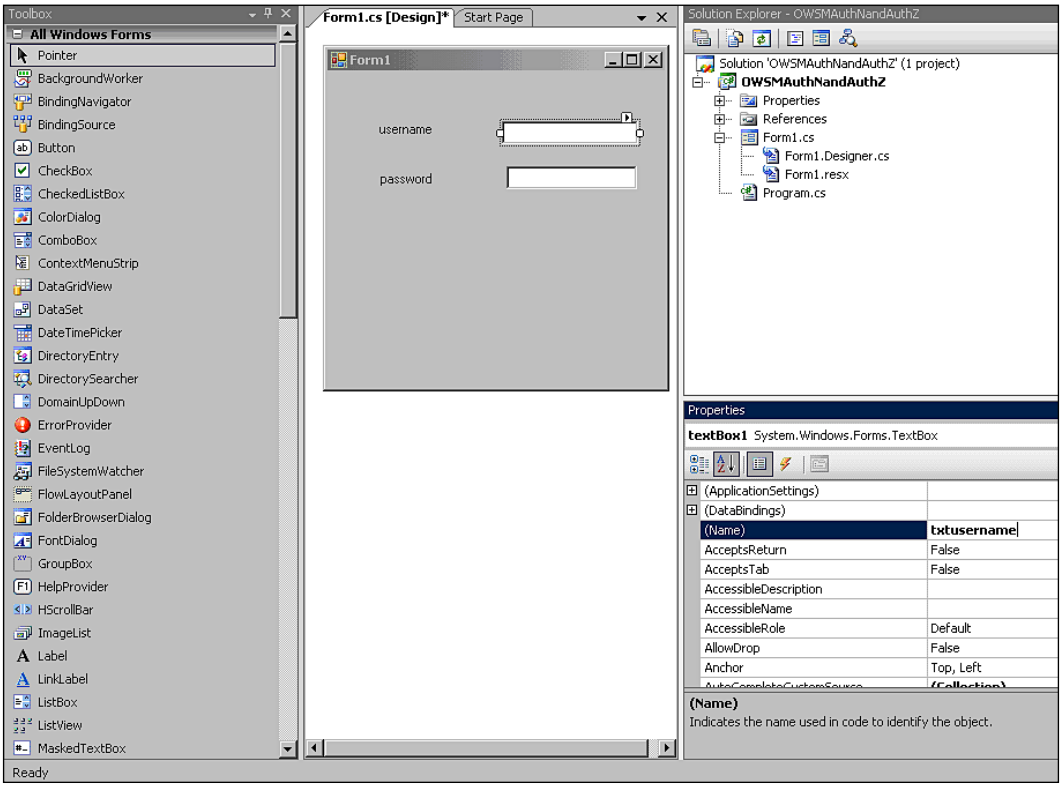
Now that the web service has been registered with Oracle WSM and the policy has been updated to make sure that only authenticated users can access the web service, let's take a look at how can we build a client application that attaches the username and password to the web service request. In this example, we will create a Windows Forms application using Microsoft Visual Studio 2005 and Microsoft Web Services Enhancements Toolkit 3.0.

In order to create the Microsoft Windows Forms project, open Visual Studio 2005 and click on **File** and then on **New Project** (refer to the following screenshot).



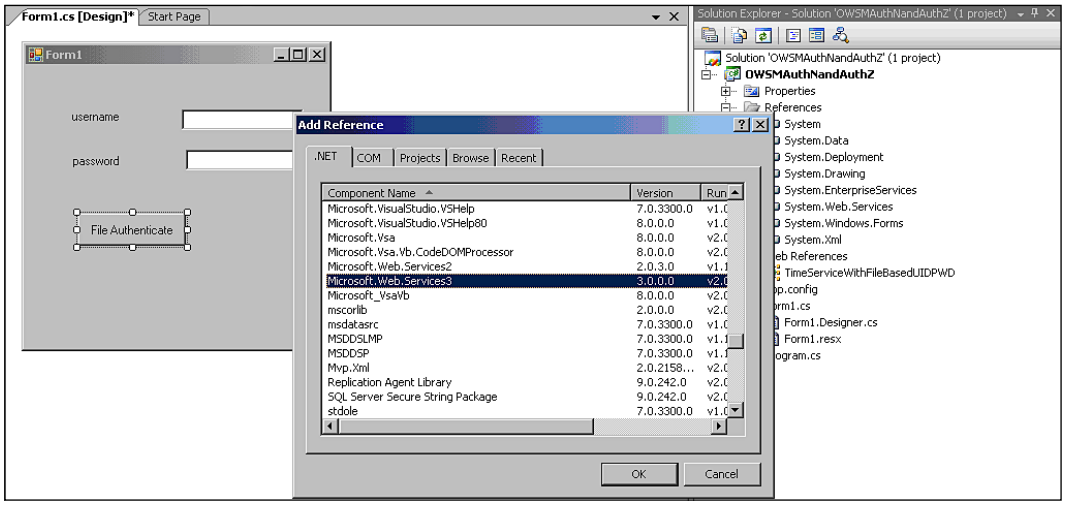
In the previous figure, the new **Windows Application** project is being opened with C# as the language.

The **Windows Application** will be accompanied by a basic windows form. You can then add text boxes, buttons, and labels to capture the **username**, **password**, error and to invoke the web service.

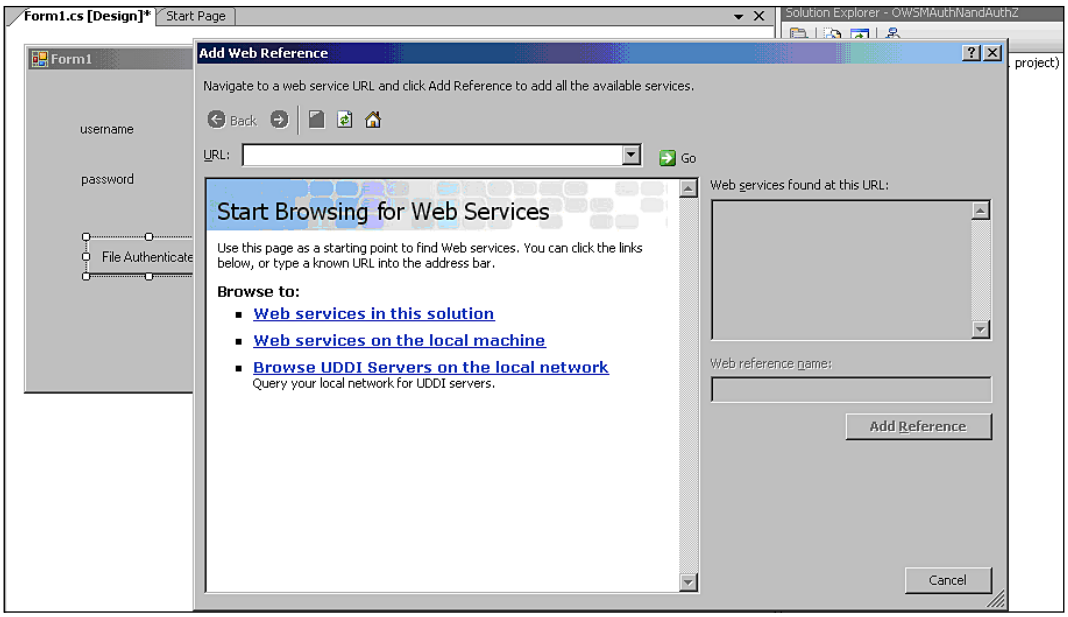


The text boxes are named as **txtusername** and **txtpassword**.

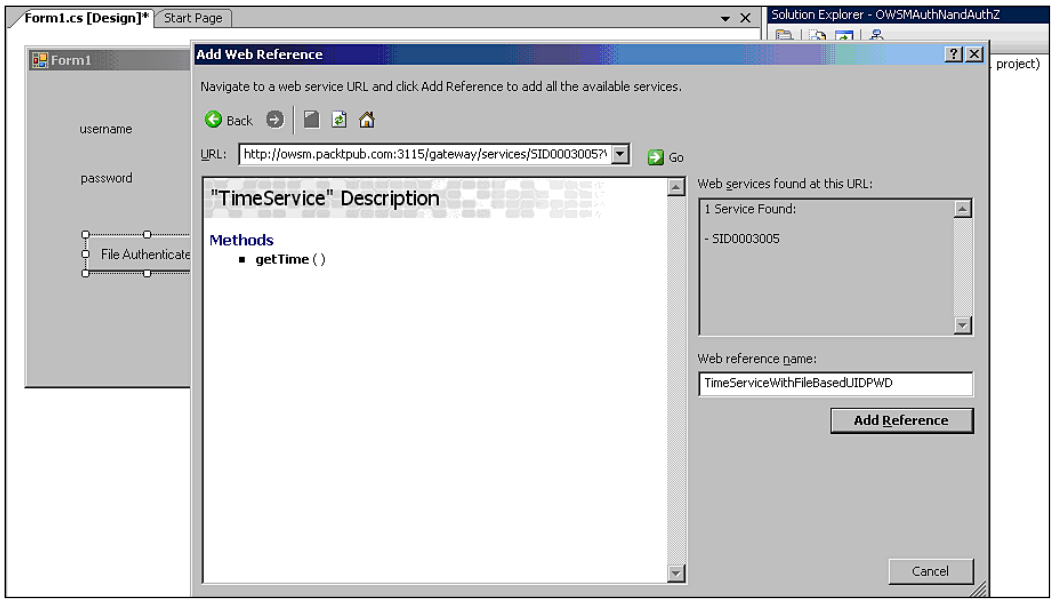
The next step is to add the web services enhancements 3.0 assembly as a reference (refer to the next screenshot).



Once **WSE 3.0** is added as reference, we can add the WSDL URL as the web reference by right-clicking on the **Web Reference** section. The following figure shows how to add WSDL as a web reference.

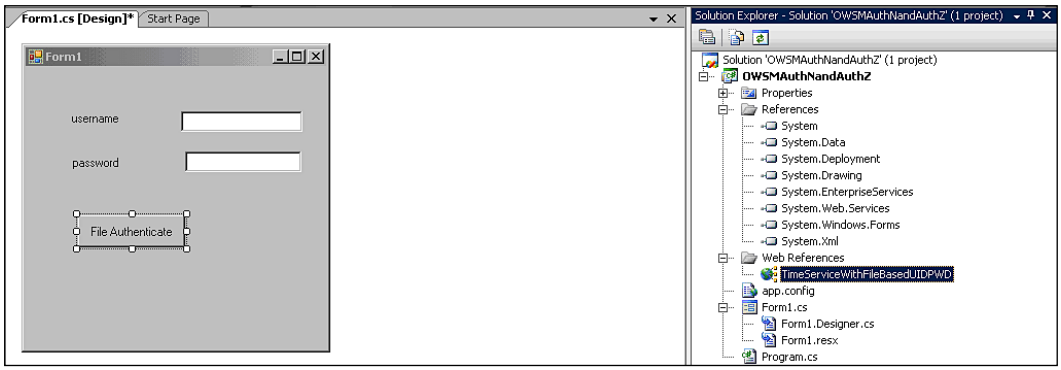


In the **URL** box, you can now type the WSDL URL. This time the URL should point to the service ID of the web service in Oracle WSM (refer to the following screen capture).



In the **Web reference name**, you can enter any name.

Now that the WSE 3.0 assembly reference has been added and the web service reference also has been added, it is time to write code to call the web service with a username and password. In the application, we have a button that can invoke the web service. We can now write code on the button click event.



Double-click on the button so that it opens up the code behind page where the web service call can be made.

In order to call the web service with a username and password, the following needs to happen:

Create an instance of the proxy class that is WSE enabled.

```
            TimeServiceWithFileBasedUIDPWD.
TimeServiceWse _timeserviceproxy = new OWSMAuthNandAuthZ.
TimeServiceWithFileBasedUIDPWD.TimeServiceWse();
```

The next step is to create the `UsernameTokenProvider` with the username and password.

```
          UsernameTokenProvider _usernametoken = new UsernameTokenPro
vider(username, password);
```

The username and password comes from the text boxes. The next step is to set the token to the proxy's client credential.

```
            //Set the username token to the Client Credential of the
                  Proxy Object_timeserviceproxy.SetClientCredential
                  (_usernametoken.GetToken());
```

Once the credentials are attached, the next step is to create the policy object and attach the `UserNameOvertransportAssertion`.

```
            //Create instance of Policy Object
            Policy webServiceClientPolicy = new Policy();
            //Create the username Over Transport Assertion
            // This assertion will send username and password
               in SOAP message webServiceClientPolicy.Assertions.
               Add(new UsernameOverTransportAssertion());
```

The policy object is then attached to the web service proxy.

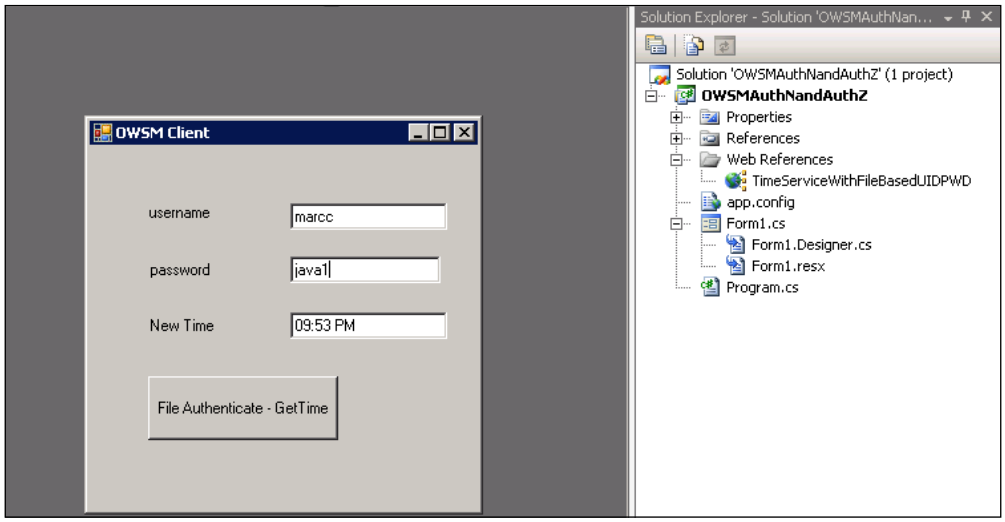
```
            // Apply the policy to the SOAP message exchange.
            _timeserviceproxy.SetPolicy(webServiceClientPolicy);
```

The complete code will look like this:

```
            //Get the username and password from screen
            string username = txtusername.Text;
            string password = txtpassword.Text;
            //Create the WSE enabled Proxy object
            TimeServiceWithFileBasedUIDPWD.
TimeServiceWse _timeserviceproxy = new OWSMAuthNandAuthZ.
TimeServiceWithFileBasedUIDPWD.TimeServiceWse();
            //Create the Username Token Provider object with username
```

```
and password
            UsernameTokenProvider _usernametoken =
                        new UsernameTokenProvider(username, password);
            //Set the username token to the Client Credential of the
              Proxy Object_timeserviceproxy.SetClientCredential
              (_usernametoken.GetToken());
            //Create instance of Policy Object
            Policy webServiceClientPolicy = new Policy();
            //Create the username Over Transport Assertion
            // This assertion will send username and password
               in SOAP message webServiceClientPolicy.Assertions.
               Add(new UsernameOverTransportAssertion());
            // Apply the policy to the SOAP message exchange.
            _timeserviceproxy.SetPolicy(webServiceClientPolicy);
            _timeserviceproxy.getTime("AnyString");
```

Now when you compile and run this project, and enter a valid username and password that exists in the directory, you will see the time as shown in the following screenshot:



Summary

In this chapter, we learnt to protect a web service with a username and password, and also wrote a client application that can attach the username and password in the web service call. In the next chapter, we will learn how to encrypt and decrypt SOAP messages.

5

Encrypting and Decrypting Messages in Oracle WSM

Encryption and decryption are key components of web services which protect the confidentiality of the data being exchanged. In order to encrypt and decrypt messages in web services, each service should know what key should be used to encrypt and to decrypt the message, the algorithm, what part of the message should be encrypted, etc. As organizations deploy more web services that require encryption or decryption, management of the key and related policies for encryption get tough to maintain. Oracle Web Services Manager addresses those issues by centrally managing and enforcing the encryption and decryption policies, including the key management. In this chapter, we will take a closer look at how we can leverage Oracle WSM to protect the confidentiality of the messages.

Overview of Encryption and Decryption

Encryption is the process of masking the data from plain text to derive at cipher text (encrypted data). Decryption is the process of deriving the plain text from the cipher text. The process of encrypting and decrypting depends on the type of encryption and the type of algorithm. There are actually two types of encryption:

Symmetric Cryptography

In symmetric encryption, plain text is converted into encrypted text using an encryption key. In order to decrypt the encrypted data, the same encryption key should be used. This is called "secret key cryptography" or "symmetric cryptography".

```
Plain Text + secret Key (Encryption Operation) = Cipher Text
Cipher Text + secret Key (Decryption Operation) = Plain Text
```

In symmetric cryptography there are various algorithms, such as DES, Triple DES, AES, Blowfish, etc.

Asymmetric Cryptography

In **Asymmetric** cryptography encryption, there are actually two keys involved. One key is used to encrypt the data and the other key is used to decrypt the data. This is also called public key cryptography. Let us consider, for example, that Alice wants to send a confidential message to Bob:

- Alice will encrypt the data using Bob's public key.
- Bob will receive the message and decrypt using his private key.

The idea is that anyone who wants to send an encrypted message to Bob can actually encrypt the data using Bob's public key.

Asymmetric cryptography is also used to digitally sign the message to ensure the integrity of the message. When Alice wants to send a message to Bob and wants to ensure that no one will tamper with the message in transit:

- Alice will digitally sign the message using Alice's private key.
- Bob will receive the message and validate the signature using Alice's public key.

When a message is digitally signed, a message digest (unique value for each message) is calculated using a digest algorithm such as MD5 or SHA1, and that value is then encrypted using Alice's private key. When Bob receives the message, he recalculates the digest again using the same algorithm and encrypts using Alice's public key, and then compares it with the one that was sent. If the values differ, we can be sure that the data was tampered in transit.

In asymmetric cryptography, there are various algorithms such as RSA, DSA, and Elliptic Curve Cryptography (ECC).

Oracle WSM and Encryption

In order to protect the confidential data in web services, the data should be encrypted. From the last two sections we understand that in order to encrypt and decrypt data, one has to know the type of encryption, the algorithm and the encryption key. In the web services scenario, where there can be multiple services in an organization, each service should know:

- What data elements are encrypted or to be encrypted
- What the encryption algorithm is
- Where the encryption key is or how to obtain one

It becomes increasingly difficult to manage the encryption and decryption policies for web services as more and more services are being deployed. **Oracle WSM** can help address these issues by centrally managing the policies where it can define the encryption and decryption processes for the incoming and outgoing **SOAP** messages.

Encryption and Decryption with Oracle WSM

In the web services scenario, the Oracle WSM gateway or the agent should be able to decrypt the incoming message and encrypt the outgoing message. In order to encrypt the outgoing message using an asymmetric algorithm, Oracle WSM has to know:

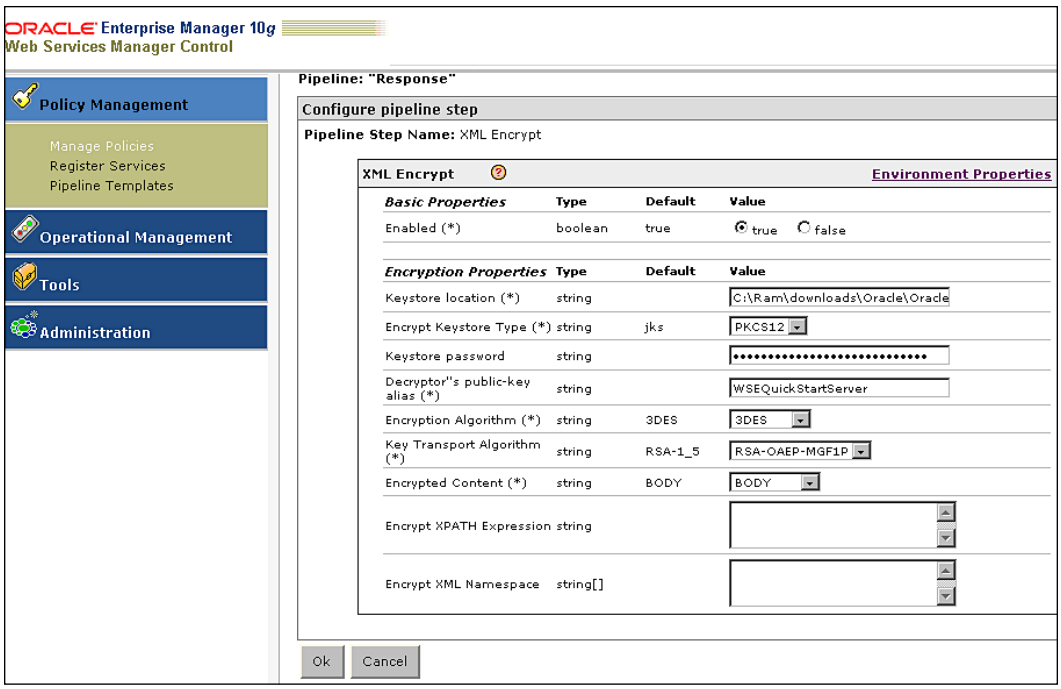
- Location of the key store that contains the public key
- Password to the key store
- Key store type—either JKS or PKCS (JKS: Java Key Store, PKCS: Public Key Crypto System. Both are used to store the public/private key)
- Alias to identify the public key
- Encryption algorithm—Triple DES, AES, etc.
- Key transport algorithm
- Encrypted content
- Namespace or XPATh info to encrypt specific data (XPath is used to query XML document to get XML element(s) and attributes)

In order to decrypt the incoming message, Oracle WSM has to know:

- Location of the key store that contains the private key
- Password to the key store
- Key store type
- Alias to the private key
- Password to access the private key

Now if you read the previous bullet points again, you will notice the key difference about the information required by Oracle WSM for encryption and decryption. They key difference is that for encryption, Oracle WSM requires the encryption algorithm and key transport algorithm, and for decryption, it requires only the private key password and does not require any information regarding the algorithm.

This is the advantage of Oracle WSM implementing the industry standards, such as the WS-security that encompasses XML encryption. The following screenshot shows the information that Oracle WSM requires to perform the encryption of the response message.



We know that the **Keystore location**, **password**, **Keystore Type**, **alias** are required to access the key store to retrieve the public key information. The **Encrypted Content** lets you choose which part of the SOAP message is to be encrypted, the **Encrypt XPATH Expression string** lets you specify the Xpath expression to specify XML elements or attributes to be encrypted. However, what is the deal with **Encryption Algorithm** and **Key Transport Algorithm**, and other parameters?

Encryption Algorithm

Encryption algorithm is used to specify the symmetric key encryption algorithm, such as Triple DES, AES, Rijndeal, etc. This is actually used to encrypt the SOAP message, and in this case, the body of the message (specified as part of the encrypted content parameter). When encrypted, the SOAP body is actually encrypted using the Triple DES algorithm.

Note: Public key encryption is usually CPU intensive and encrypting large XML documents using public key will affect the performance of the system. Hence it is not uncommon to encrypt the SOAP body using a secret key (symmetric cryptography) and then encrypt the "key" with the public key.

Key Transport Algorithm

The previous section described that the encryption algorithm parameter in Oracle WSM is used to encrypt the SOAP message. In order to decrypt the message, the recipient should know the encryption key. The obvious question then is how is the encryption key information transported?

The encryption key can be transported outside of Oracle WSM and the client can decrypt using their own tools or the encryption key itself can be encrypted. If you are wondering how encrypting the encryption key is going to help with transporting the encryption key, the working when Oracle WSM is configured to encrypt the response message is as follows:

- SOAP Message (in this case, Body) is encrypted using triple DES algorithm
- Encryption key is generated at run time
- Using the specified key transport algorithm, encryption key is encrypted with the public key
- Encrypted encryption key is transported with SOAP message (`EncryptedKey` element)

Internal Working of the XML Encrypt Policy Step

Now that we know about the XML encrypt policy step and the options that are available within the policy step, let's take a look at how it internally works when the SOAP message is transferred in an encrypted format. This will help us understand the importance of each option and how it is related.

The sample XML message show how the SOAP body is encrypted.

```
<soap:Body>
      <xenc:EncryptedData xmlns="http://www.w3.org/2001/04/
       xmlenc#" xmlns:xenc="http://www.w3.org/2001/04/
       xmlenc#" Type="http://www.w3.org/2001/04/xmlenc#Content"
       Id="_63BQq2iUOIn6m00ZkQ2F0A22">
            <xenc:EncryptionMethod Algorithm=
             "http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
            <xenc:CipherData>
                                <xenc:CipherValue>**EncryptedDa
                                 taVneQADWk=</xenc:CipherValue>
            </xenc:CipherData>
      </xenc:EncryptedData>
</soap:Body>
```

If you look at the message, the `EncryptedData` element has a client element called `EncryptionMethod` that specifies the algorithm as triple DES. The `CipherValue` element contains the encrypted SOAP body. The encryption key is encrypted and transported along with the SOAP message, and it is referenced by `EncryptedKey` and is part of the SOAP header. The following message shows the SOAP header value that contains the `EncryptedKey` element:

```
      <xenc:EncryptedKey xmlns=
      "http://www.w3.org/2001/04/xmlenc#" xmlns:
       xenc="http://www.w3.org/2001/04/xmlenc#">
            <xenc:EncryptionMethod Algorithm=
             "http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p">
               <dsig:DigestMethod Algorithm=
                "http://www.w3.org/2000/09/xmldsig#sha1" xmlns:
                dsig="http://www.w3.org/2000/09/xmldsig#"/>
                  </xenc:EncryptionMethod>
                  <dsig:KeyInfo xmlns:dsig=
                  "http://www.w3.org/2000/09/xmldsig#">
                     <wsse:SecurityTokenReference
                     xmlns="http://docs.oasis-open.org/
                     wss/2004/01/oasis-200401-wss-wssecurity-
                     secext-1.0.xsd">
                       <wsse:Reference URI="#a2XD1JG7keBavISVs
                       j2HxLw22" ValueType="http://docs.oasis
                       -open.org/wss/2004/01/oasis-200401-wss-
                       x509-token-profile-1.0#X509v3"/>
                         </wsse:SecurityTokenReference>
                  </dsig:KeyInfo>
                  <xenc:CipherData>
                     <xenc:CipherValue>**encrypted key value/
                     SDqW4WKY15/7w=</xenc:CipherValue>
```

```
                    </xenc:CipherData>
                    <xenc:ReferenceList>
                         <xenc:DataReference URI=
                         "#_63BQq2iUOIn6m00ZkQ2F0A22"/>
                    </xenc:ReferenceList>
               </xenc:EncryptedKey>
```

In the previous sample message, the `EncryptedKey` element represents information about the encrypted encryption key and is a reference to what data is encrypted using that encryption key. In other words, the `EncryptedKey` element contains:

- `EncryptionMethod` element to describe what algorithm is used to encrypt the encryption key.
- `CipherData` and `CipherValue` elements contain the actual encrypted data (encrypted encryption key)
- `KeyInfo` element gives information about the key, and in this case, it's public key.

So far the elements above describe the key, the algorithm, and the location of the encrypted encryption key. The `DataReference` element, a part of `ReferenceList` element, contains a list of references that point to the **EncryptedData** element, which contains the actual data that is encrypted.

In a nutshell, when Oracle WSM encrypts the portion of the SOAP message (in this case, SOAP body) here is what happens internally:

- Symmetric encryption key is generated at random.
- SOAP body is encrypted using triple DES algorithm with the encryption key.
- The encrypted data (`CipherValue`) and algorithm (`EncryptionMethod`) are represented as a part of the `EncryptedData` element.
- `EncryptedData` is identified with `Id` attribute.
- The encryption key is then encrypted using RSA OAEP algorithm (Optimal Asymmetric Encryption Padding, i.e. message is OAEP encoded and then encrypted with RSA algorithm)
- The **encrypted encryption key** is represented by the **EncryptedKey** element along with the algorithm, the key information and the cipher text (encrypted value of encryption key).
- The relationship between the `EncrypedData` element and `EncryptedKey` is established by the `DataReference` element that is a part of the `EncryptedKey`.

Once the client application receives the SOAP message, it can decrypt the data using the information that is available within the XML message. During the decryption process, the client application, that is aware of WS-security standards, can easily understand from the XML message that:

- The encrypted data in the `EncryptedData` element is encrypted using triple DES.
- The key information is available within the `EncryptedKey` element.

Once the client application understands the various elements, it will then:

- Decrypt the encryption key using its private key (from the `EncryptedKey` element).
- Use the encryption key to then decrypt the SOAP message.

In the next section, we will explore how to leverage Oracle WSM to encrypt and decrypt messages with a sample application.

Oracle WSM Sample Application Overview

In order to demonstrate the capability of encryption and decryption within Oracle WSM we will deploy a web service within Oracle WSM and configure the policy to protect the web service. The idea is to demonstrate both encryption and decryption using Oracle WSM, so in our sample we will have Oracle WSM:

- Enforce that the incoming SOAP message should be encrypted.
- Decrypt the incoming SOAP message (request).
- Enforce that the outgoing SOAP message should be encrypted.
- Encrypt the outgoing SOAP message (response).

In order to have a complete demo sample, we will also a develop a client application using Microsoft .NET, and the client application will:

- Encrypt the outgoing message.
- Decrypt the incoming message.

Since the web service development is outside the scope of this book, we will leverage the existing web service that comes with Oracle WSM installation, the time service. The time service takes the format as an input string, but it doesn't really matter what data is entered, and you will get a response with actual time. The sample application will demonstrate how to use:

- Microsoft .NET application to encrypt the SOAP request (i.e. body of the SOAP message, the format string).
- Oracle WSM to decrypt incoming SOAP message.
- Oracle WSM to encrypt the response message that contains the actual time value.
- Microsoft .NET application to decrypt the SOAP response (the actual time in the SOAP body).
- Both the client and Oracle WSM will use the WSEQuickStartServer key that ships with Microsoft WSE 3.0.

In the next section, we will explore how to configure the Oracle WSM policy to encrypt and decrypt messages in detail.

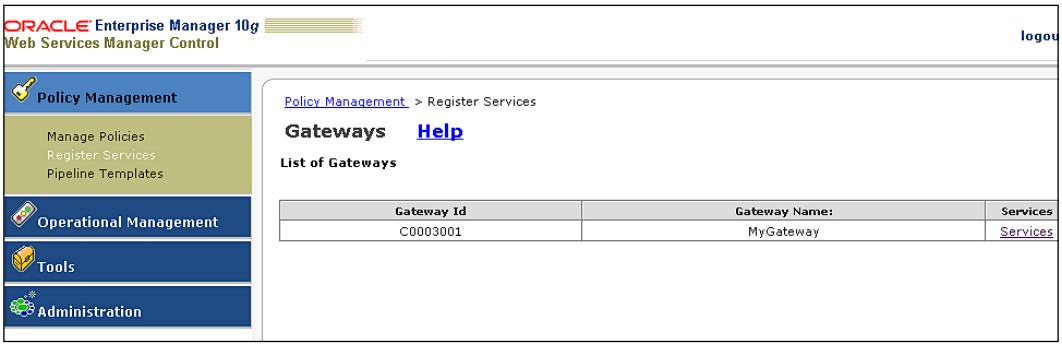
Oracle WSM Encryption and Decryption Policy

In order to protect a web service within Oracle WSM, the first step is to register the web service within Oracle WSM and then edit the policy associated with it. The following steps describe how to register the service.

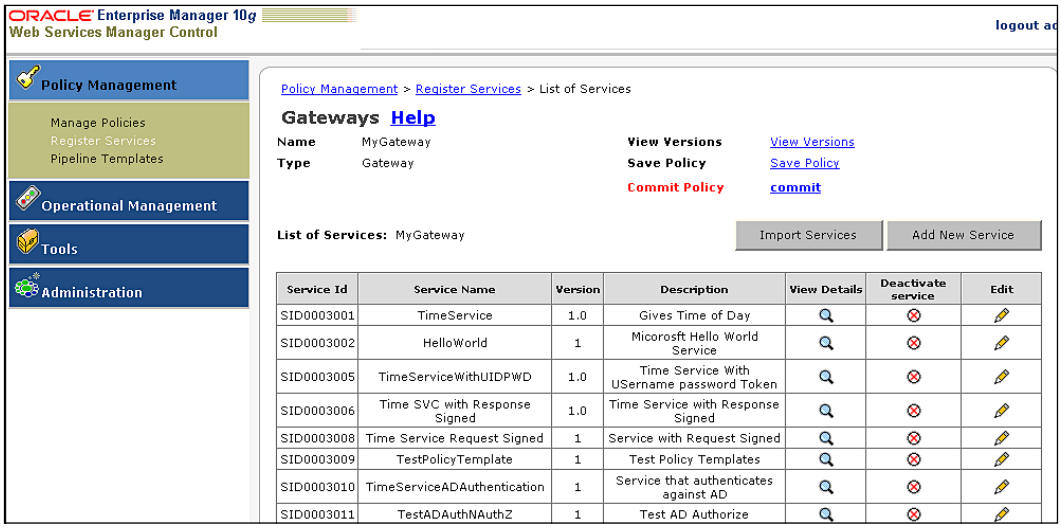
Login to Oracle Web Service Manager Console at:

```
http://owsm.packtpub.com:3115/ccore
```

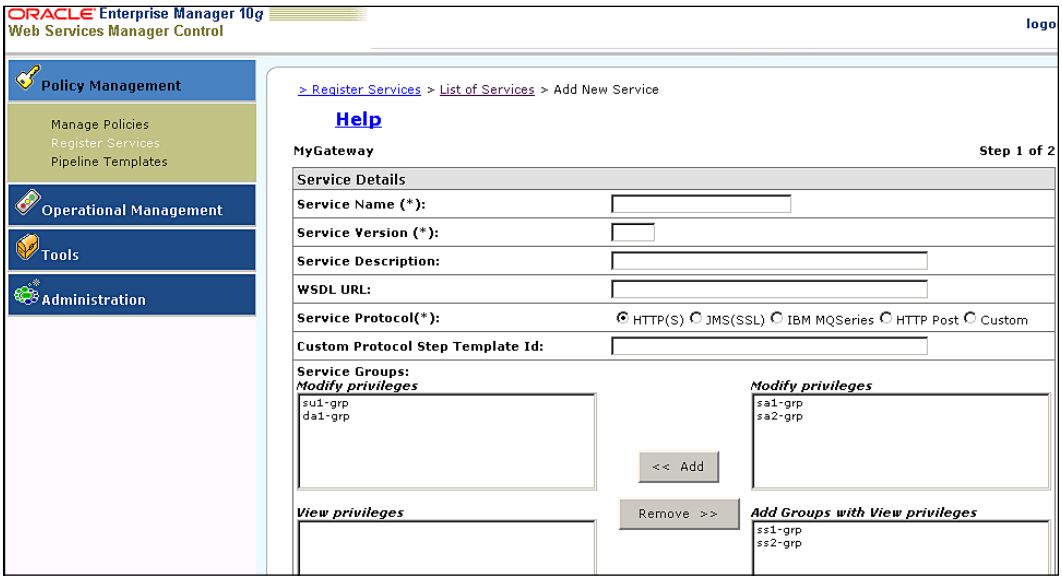
Click on **Policy Management** and then **Register Services**. You will see the list of **Gateways** that are available, as shown in the following screenshot.



On the right side of the screen, if you click on the **Services** hyperlink, you will see the list of registered services (refer to the following screenshot).



In order to add a new service, click on **Add New Service** from the right side panel.



The previous screen capture describes the details that can be added while adding the new service. The * after each label makes those fields mandatory. The screen asks for typical information such as:

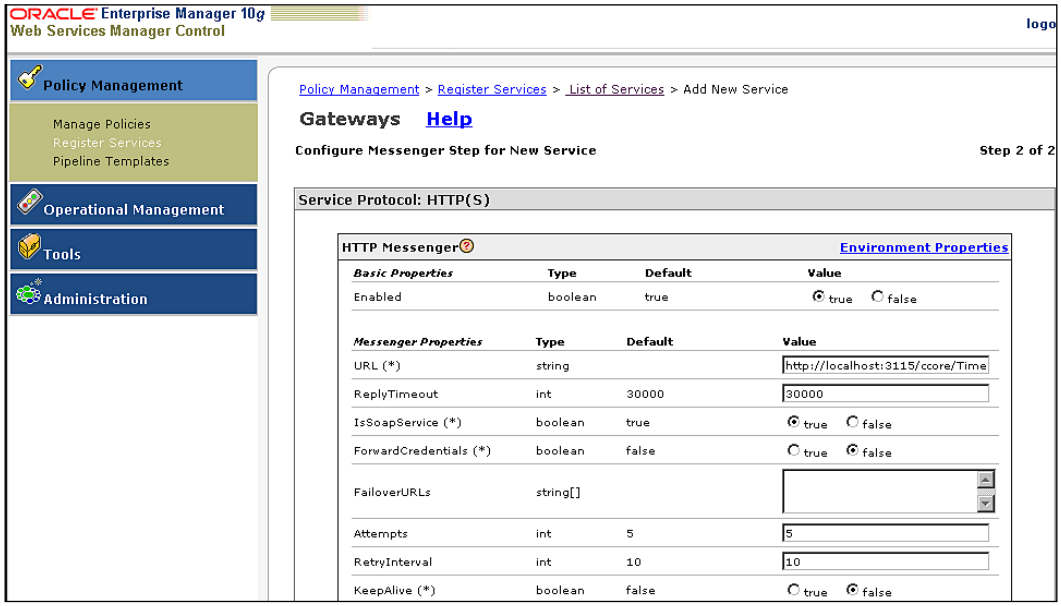
- Name of the service
- Version of the service
- Any description of the service
- WSDL URL of the service
- Protocol in which the service will accept messages

It also asks for additional information such as **Service Groups**, the Oracle WSM groups that has previleges to view and edit.

In our example, we are registering the time service that will authenticate users against active directory. The time service is registered with the following information:

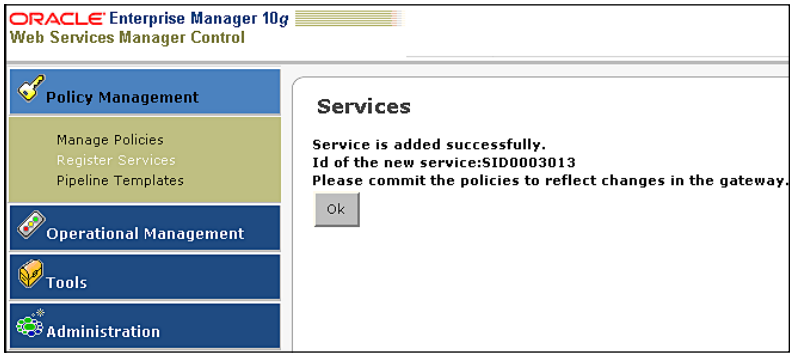
- Name of service: EncryptionDecryptionSample
- Service Version: 1
- Service Description: Encrypt and decrypt SOAP message
- WSDL URL: `http://owsm.packtpub.com:3115/ccore/TimeService.wsdl`
- Service Protocol: HTTP(S)
- Service Groups: Select the default that has full permissions

Once the information is filled out, click **Next** on **New Service** registration. This will take you to the next screen which will display the actual URL of the service (see the following screenshot).



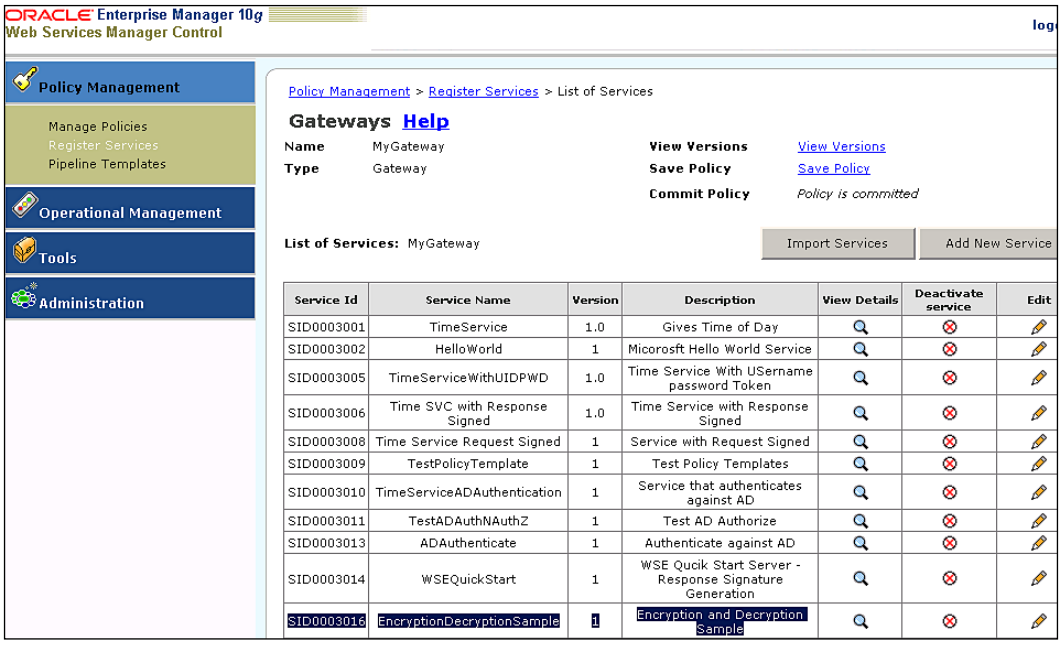
The URL in this page comes from the WSDL which contains the actual web service code (i.e. in this case, the JSP page). You just have to make sure that the service is enabled and also check if it is Soap service or not.

You can then click **Finish** to register the service. Once you click **Finish**, the Oracle WSM internally generates a new service ID and now the client applications can use that service ID to communicate.



The previous screenshot show that Oracle WSM registered the time service and created a new Service ID.

Click **OK** to get back to the main screen that lists all the services.

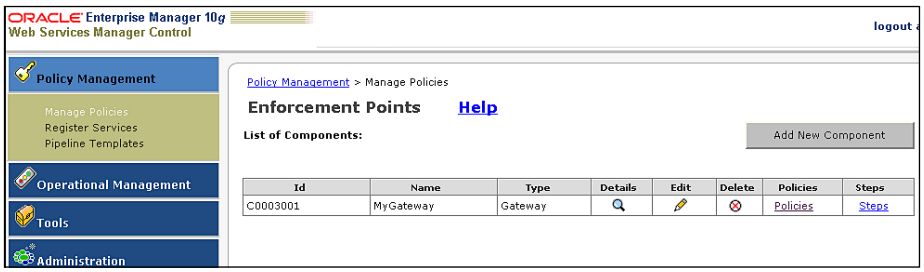


The previous screenshot shows all the services. The highlighted one is the **EncryptDecrypt** service that was just registered.

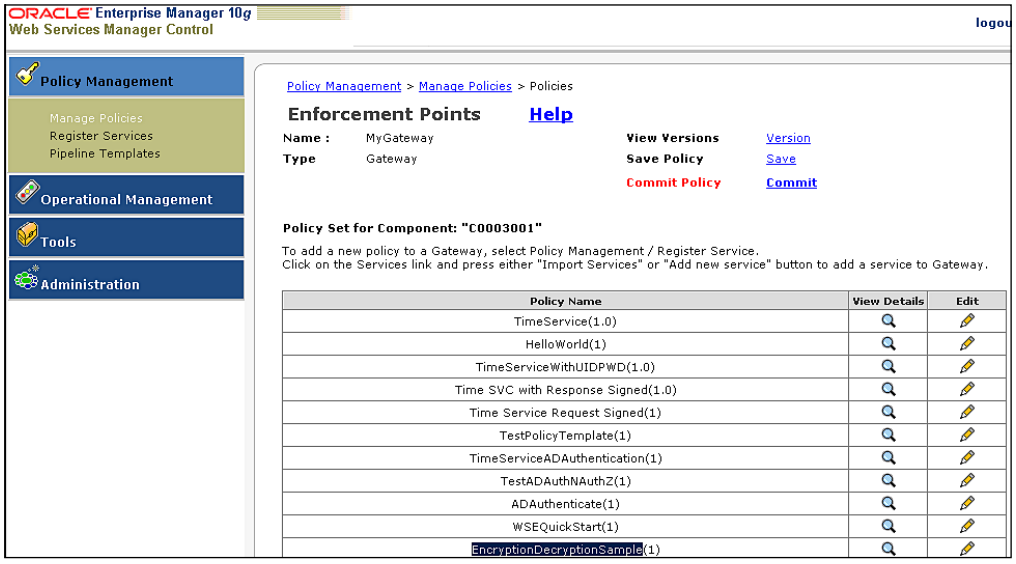
We have only added the new service, but it hasn't been committed yet. The previous figure shows the **Commit Policy** in red and you can now click on that to commit the policy. Policy steps are not saved until it is actually committed.

Creating the Security Policy

In the previous section, we learnt how to register the service and commit the policy. Once a service is registered and committed, it is associated with a default policy without any security. In order to view the policy, you can click on **Policy Management** and then **Manage Policies**. This will bring you to the screen with the **Gateway** information and a hyperlink for **Policies** (see the following screen capture).

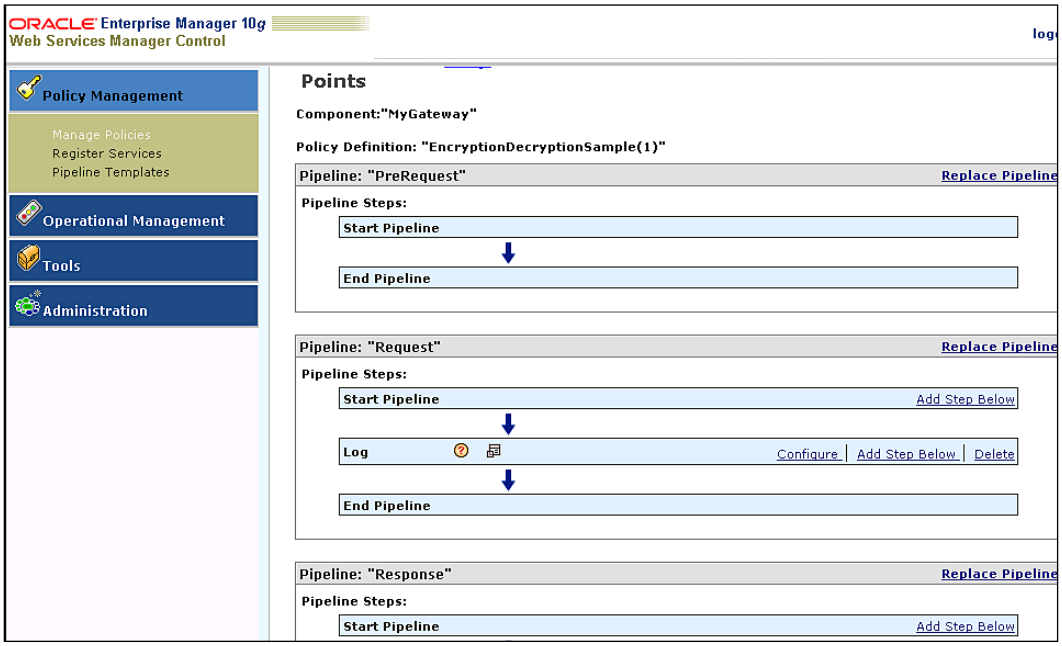


You can then click on **Policies** to see all the policies, and you will see the **`EncryptDecrypt`** policy that is created by default (see the following screenshot).

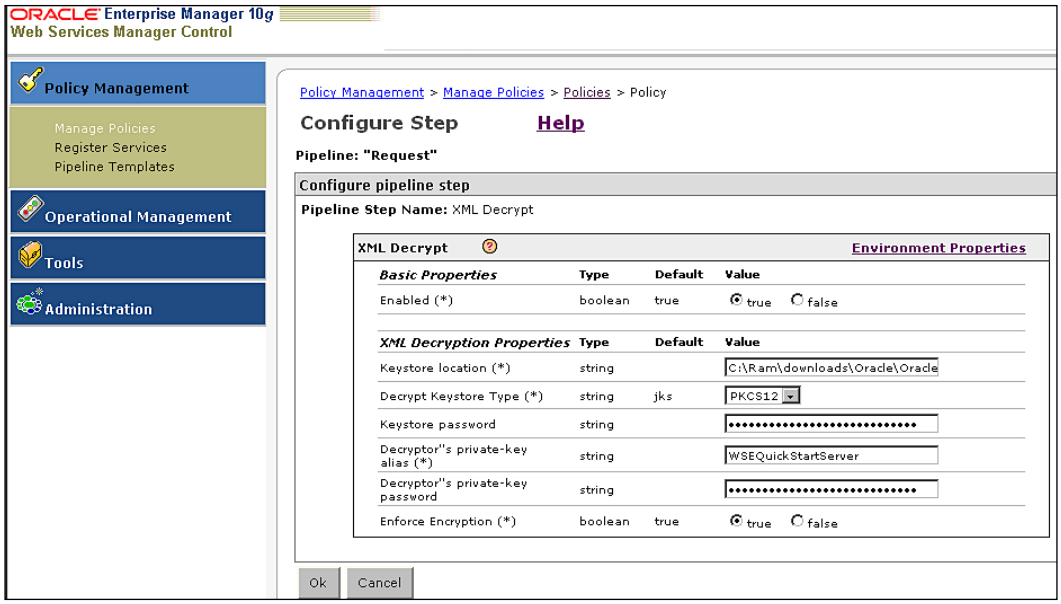


At this point, a dummy policy is attached to the newly registered EncryptDecrypt service without any security. Any application can now call this web service without the request or response message being encrypted.

In order to enforce the encryption and decryption policy, you have to edit this policy. This is done to make sure that the Oracle WSM checks the SOAP message request to enforce encryption, and that the response message is encrypted. When you click on the **Edit** button on the policy, the default policy steps are described, as shown in the following screenshot.



In order to decrypt the SOAP Request, the **Request** pipeline should be modified to include the **XML Decrypt** policy step. In order to add the new policy step, click on **Add Step Below** under the **Log** step and select the **XML Decrypt** policy step from the dropdown.



The previous figure shows the **XML Decrypt** policy step that describes:

- Location of the key store, i.e. key store location that points to PKCS 12 certificate file
- Keystore Type i.e. PKCS12
- Keystore password i.e. in this case it is `test123`
- Private Key Alias, i.e WSEQucikStartServer
- Password to access the private key i.e. in this case `test123`
- Enforce Encryption option is **true** to make sure that the incoming message is encrypted

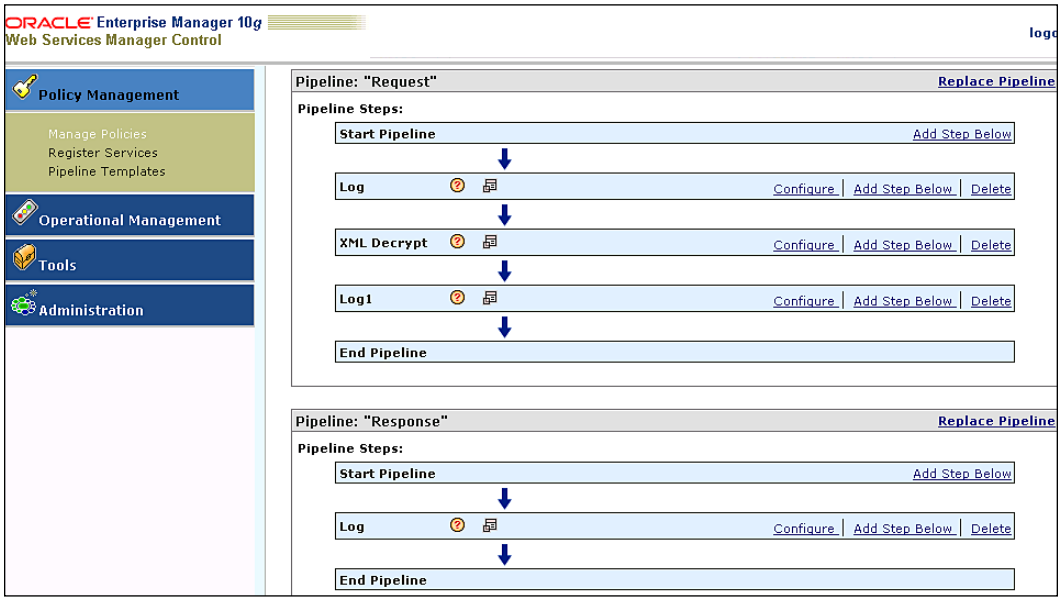
Note: The **WSEQuickStartServer** key that ships with Microsoft WSE does not have an alias (i.e. friendly name) and it is not straightforward to convert to a **PKCS12** key store. Here are the steps to create the **PKCS12**:

- Import the certificate into Microsoft certificate key store by just double clicking on the certificate file and follow the prompts
- Modify to enter the friendly name, i.e. alias
- Click **Export** to export with private key and save to a location
- Install Firefox browser
- Import the certificate from the **Advanced Security** tab of Firefox
- Export the certificate and save as PKCS12 file.

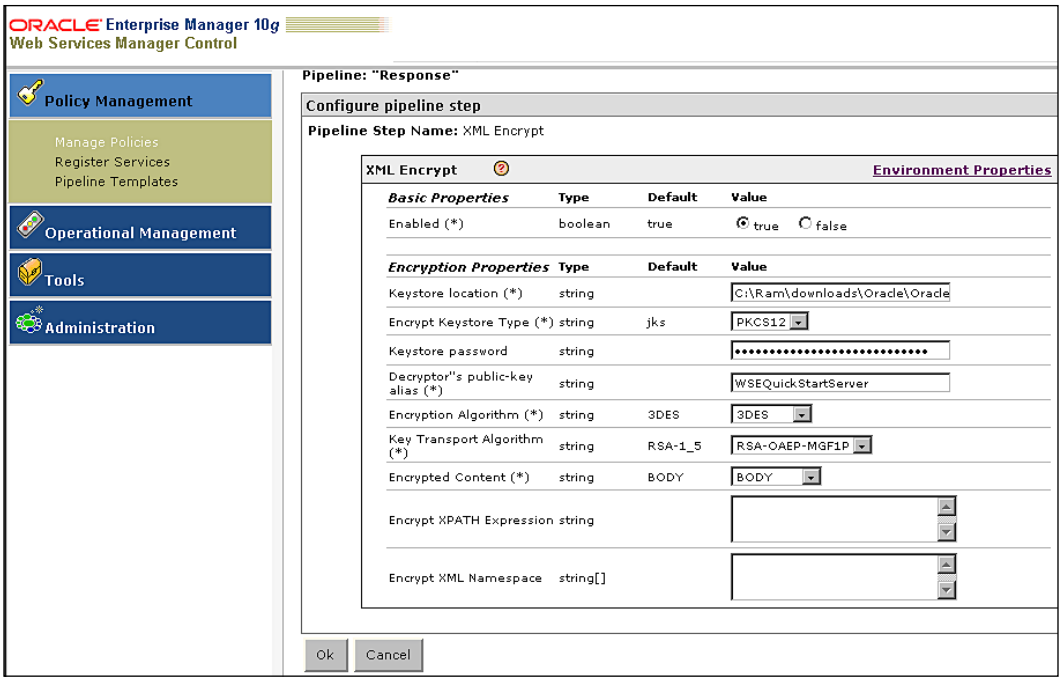
Once all the information is entered at the **XML Decrypt** policy step, click **OK** to return to the **Policy** screen.

Now that we have configured the **Request** pipeline to enforce encryption and can also decrypt the message, the next step is to configure encrypting the response message.

From the policy definition of **Encrypt Decrypt**, we should add steps to encrypt the response message.



The response pipeline has the **Log** step configured, and now we have to add the XML encrypt policy step under the **Log** step. We can now click on the **Add step below** under the **Log** the step of response pipeline and select **XML Encrypt** from the list of policy steps. The **XML Encrypt** policy step would look like the following screenshot:



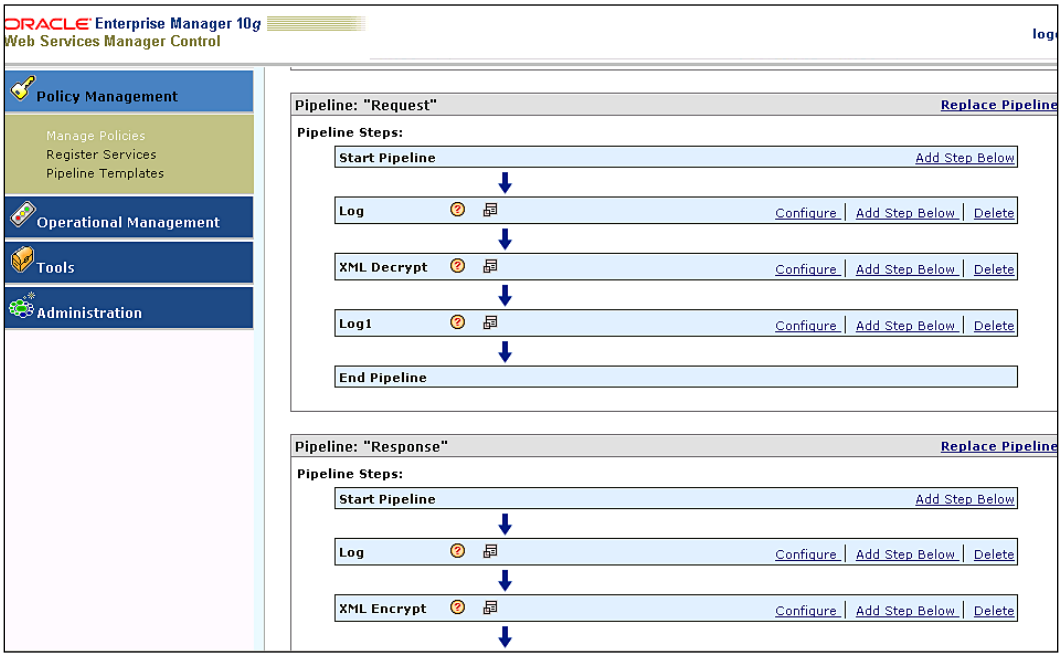
The previous figure describes the details of the XML Encrypt policy step. Before we enter the details, we should know:

- What algorithm are we going to use to encrypt the data?
- What algorithm are we going to use to encrypt the encryptionkey?

In this sample, we have configured to use the same **WSEQuickStartServer PKCS12** key store, and we have selected triple DES for the encryption of data and RSA OAEP to encrypt the encryption key.

Note: RSA OAEP is chosen because Microsoft WSE 3.0 by default can only understand RSA OAEP.

Once the details are entered, you can click **OK** and then save the policy. Once the policy is committed, the policy steps for EncryptDecrypt would look like the following screenshot.



In the previous figure, you can see the **Log** steps configured twice for both request and Response pipelines. This is to capture the data before and after request and response so that you can take a look at how the data looked before the policy step and how it looks after the policy step.

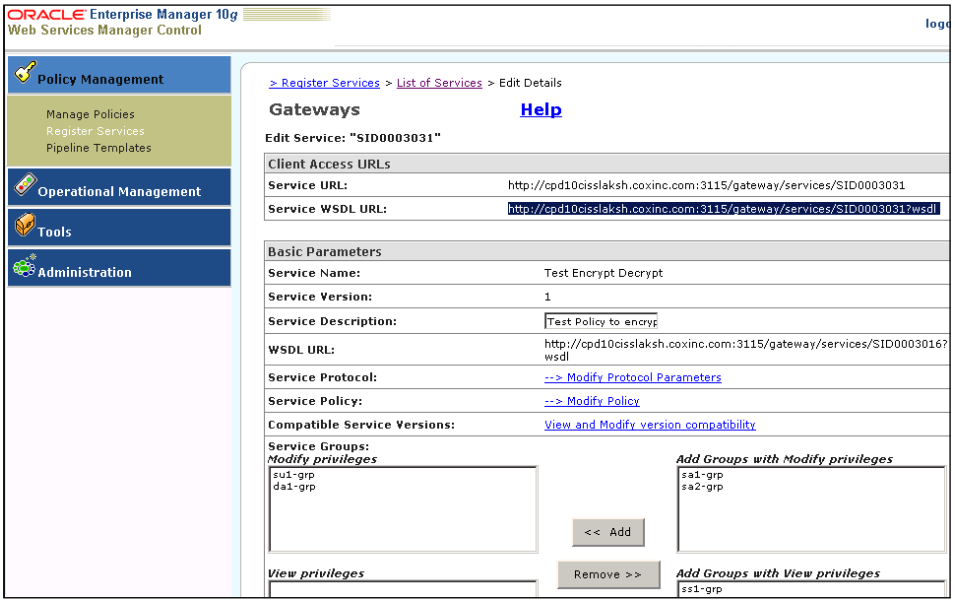
Now that we have configured the Oracle WSM to decrypt the SOAP request and encrypt the SOAP response, we should create the client application which can actually encrypt the SOAP request and decrypt the response to show the actual time. We will take a look at how to use Oracle WSM test page and Microsoft .NET to test the web service policy where the request message is encrypted and the response message is decrypted to view the time.

Oracle WSM Test Page as Client Application

The EncryptDecrypt example policy that was described in the earlier section requires that the web service request should be encrypted. The web service URL for the time service after it is registered within Oracle WSM (where the Encrypt Decrypt policy was applied) is `http://owsm.packtpub.com:3115/gateway/services/SID0003016?WSDL`. Since the web service policy requires that the SOAP request should be encrypted, you cannot really use the test page from Oracle WSM where the request message is encrypted. However, you can still use Oracle WSM to simulate the behavior. Here is how it works:

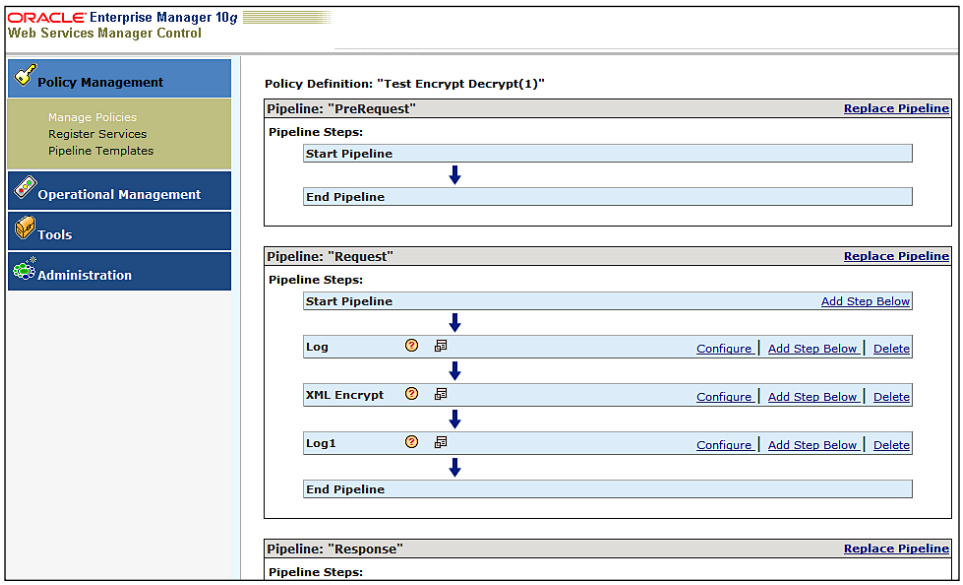
- You take the WSDL of the web service that you just registered within Oracle WSM (SID0003016?WSDL).
- Register that WSDL in Oracle WSM.
- Now a new WSDL URL is generated by Oracle WSM (SID0003031?WSDL).
- Define the policy in such a way that you include the XML Encrypt policy step in the request pipeline.
- Now you invoke the newly generated WSDL from the test page.
- SOAP request to our original web service (SID0003016?WSDL) is now encrypted.

Let's go through the steps to test our EncryptDecrypt policy. The first step is to register the SID0003016 web service within Oracle WSM and the new service URL will be created, as shown in the following screenshot. (Note: you would follow the normal process of clicking on **Services** and enter SID0003016?WSDL)



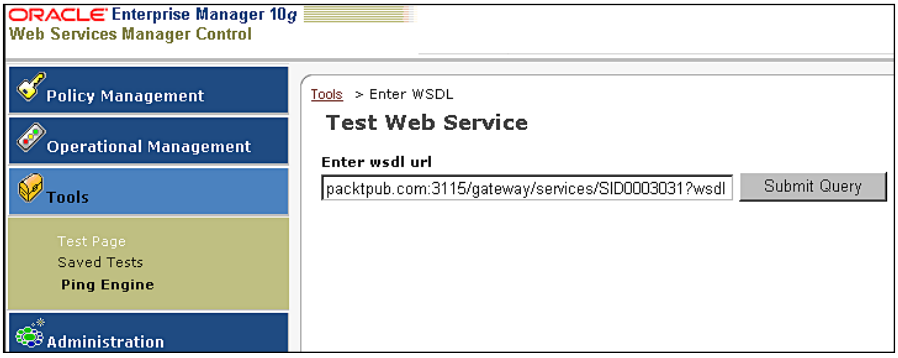
Once the web service is registered, it generates a new WSDL which is identified as `http://owsm.packtpub.com:3115/gateway/services/SID0003031?WSDL` and the service is registered as **Test Encrypt Decrypt**.

Now that the service is registered, you can edit the policy to define the request pipeline. Since this web service will simply call our original web service, we will implement the **XML Encrypt** policy in the **Request** pipeline to encrypt the web service request, as shown in the next screen capture.

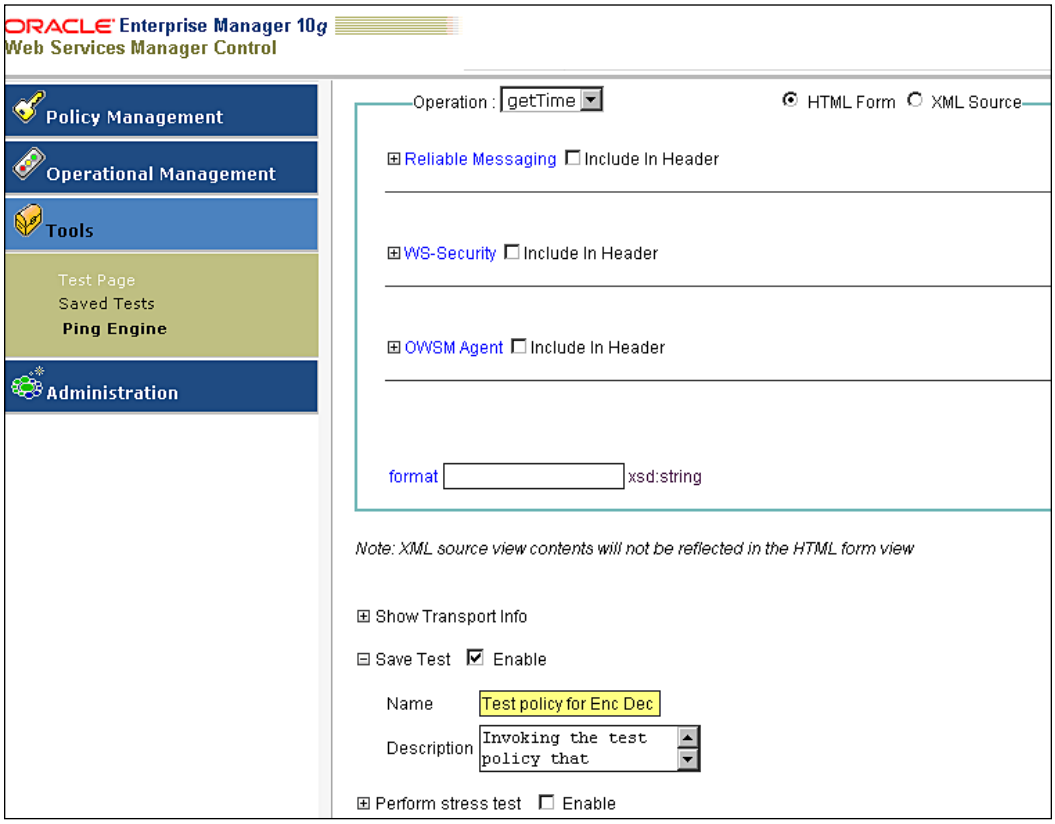


Now that the policy is defined and committed, we can now test the web service from the test page.

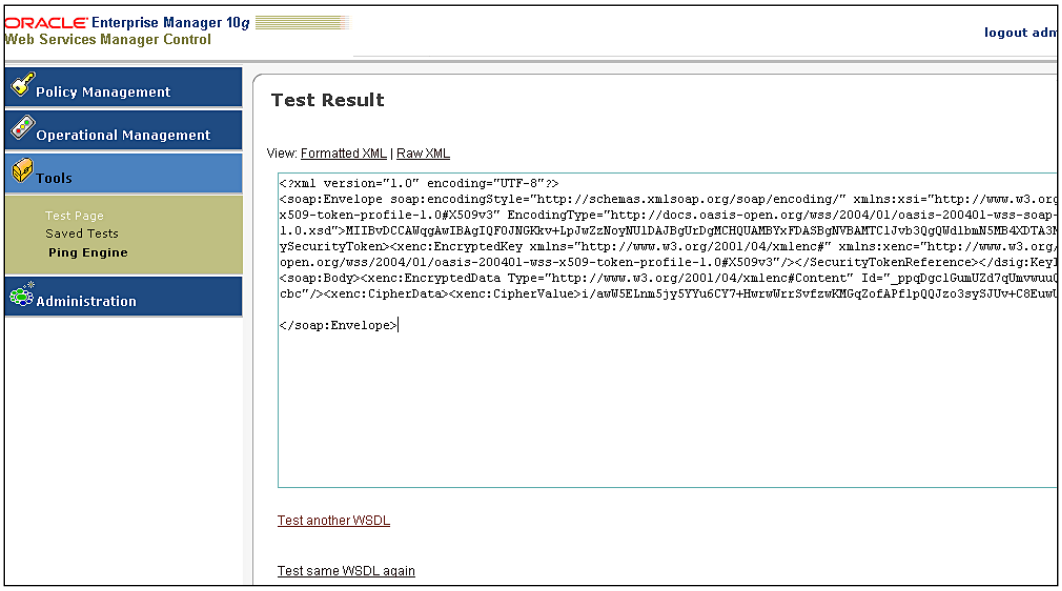
In the test page, you can enter the location of the WSDL as `http://owsm.packtpub.com:3115/gateway/services/SID0003031?WSDL` (see the following screenshot).



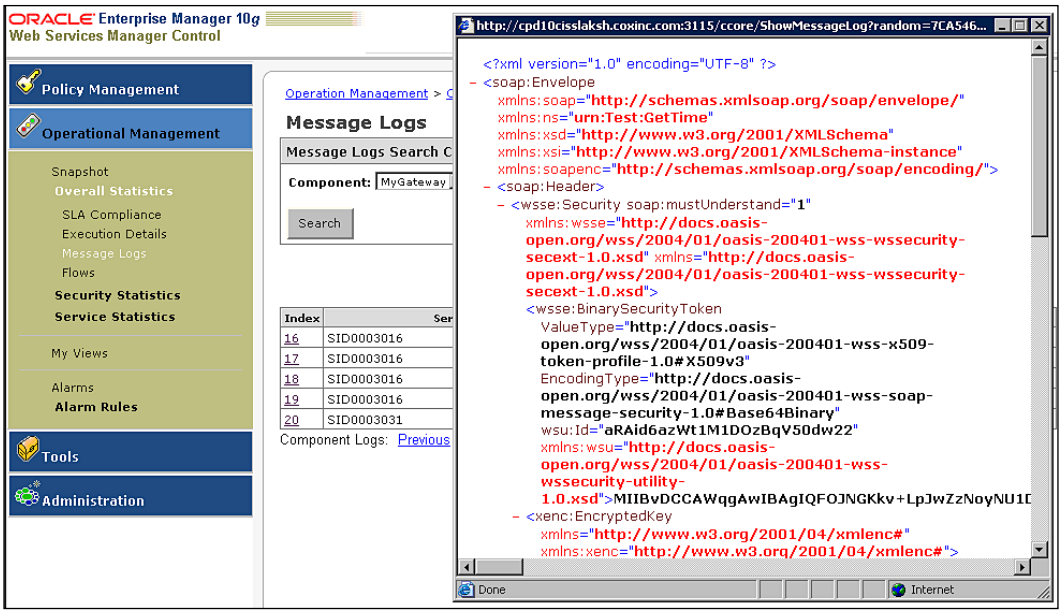
Now you can click on **Submit Query** and then check the **Save Test** page button and enter the **Description** to save the test page (refer to the following screen capture).



You can now click on the **Invoke** button, which will invoke the SID0003031 web service. The policy defined in SID0003031 will encrypt the SOAP request for SID0003016 and the policy for SID0003016 will first decrypt the SOAP request and then will generate the encrypted SOAP response message, as shown in the following screenshot.



You can actually verify that the request to SID0003016 (original web service) is encrypted by looking at the **Message Logs** under **Operational Management** as shown in the following screen capture.

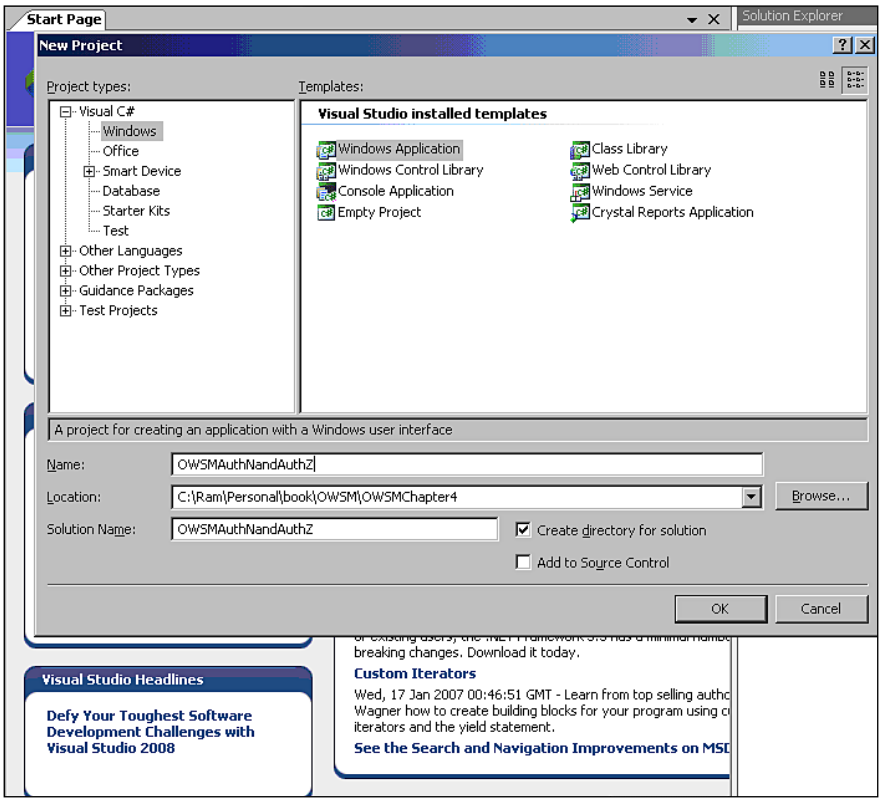


We are using Oracle WSM Policy to actually test our previous policy, but it also brings up a point—that you can implement security for any outgoing web service (especially to external organizations or applications) within Oracle WSM Gateway without having to install and configure any client agents. This design also brings in the following advantages:

- You can virtualize the web services.
- You can centralize the security for all external web services interaction.
- You can even implement identity propagation and translation (security token service):
    - Example: If you want to invoke Amazon web service which requires a SAML token, you can register that within Oracle WSM and in the policy. First validate the username and password, and then attach SAML or any other token on the request pipeline.

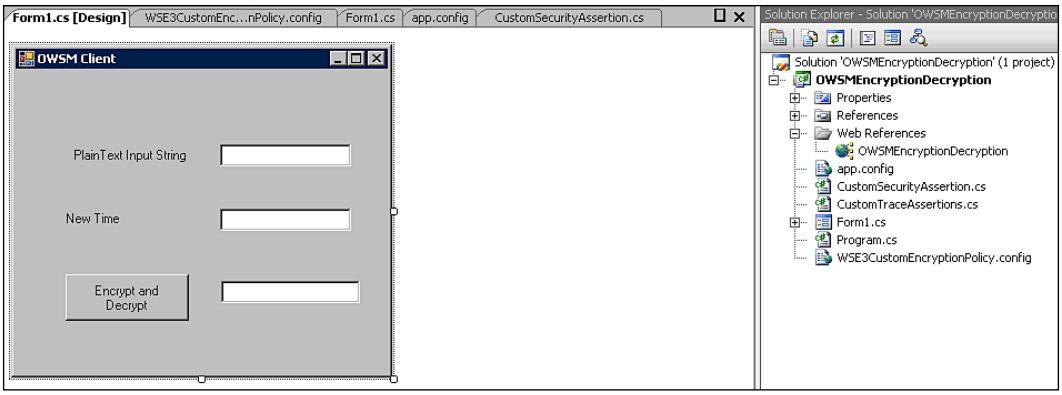
Microsoft .NET Client Application

The client application is a Windows Forms application written using Microsoft Visual Studio 2005 with Microsoft WSE 3.0. The first step in creating client application is to create a Windows Forms application called OWSMEcnryptionDecryption. The following screenshot shows how to create the Windows Forms application (as described in last chapter).



Once you click **OK**, the Windows Forms application is created with a default form.

Since we are going to invoke the time web service with input data, we will add a text box to enter any random input data (refer to the following screen capture) and a button to invoke the web service. We will also add another text box to show error messages, if any.



Now we should add the Microsoft WSE 3.0 as a reference before we can create the proxy class. You can add the WSE reference by selecting **Add Reference** from the project and then selecting **Microsoft.Web.Services3**.

Now that WSE 3.0 is added, we can then add a web reference to the web service to create the proxy object. It can be added by right-clicking on the **Add Web Reference** and then entering the WSDL URL, and then clicking **OK** to generate the proxy.

So far we have:

- Created the Windows Forms application
- Added reference to WSE 3.0
- Created the web service proxy

The next step is to configure and write code to perform the encryption and decryption. Though it is beyond the scope of this book to explain in detail how to configure policy and write custom policy assertions, we will take a high level overview of how Microsoft .NET and WSE 3.0 can be used to encrypt and decrypt SOAP messages.

With Microsoft .NET and WSE 3.0, the web services security can be applied through:

- Policy configuration for certain frequently-used security configurations, or;
- Custom assertion in combination with the policy to perform all other security operations.

Since our example focuses only on encrypting the request and decrypting the response, we will have to write a custom policy assertion. This is done so that the SOAP messages can be interpreted both during request and response cycle to perform encryption or decryption. The next step is to write the custom assertion that will:

- Encrypt the outgoing message
- Decrypt the incoming message

At a high level, the custom assertion inherits from `SecurityPolicyAssertion` class, and has methods such as `CreateClientOutputFilter` and `CreateClientInputFilter` which can be overrriden to perform any operation at the client side. In our sample, we have created two new classes called `CustomSecurityClientInputFilter` and `CustomSecurityClientOutputFilter` to perform any data processing on a request or response.

The `CustomSecurityClientOutputFilter` is derived from `SendSecurityFilter` and has a method called `SecureMessage` which can be overridden to perform the encryption.

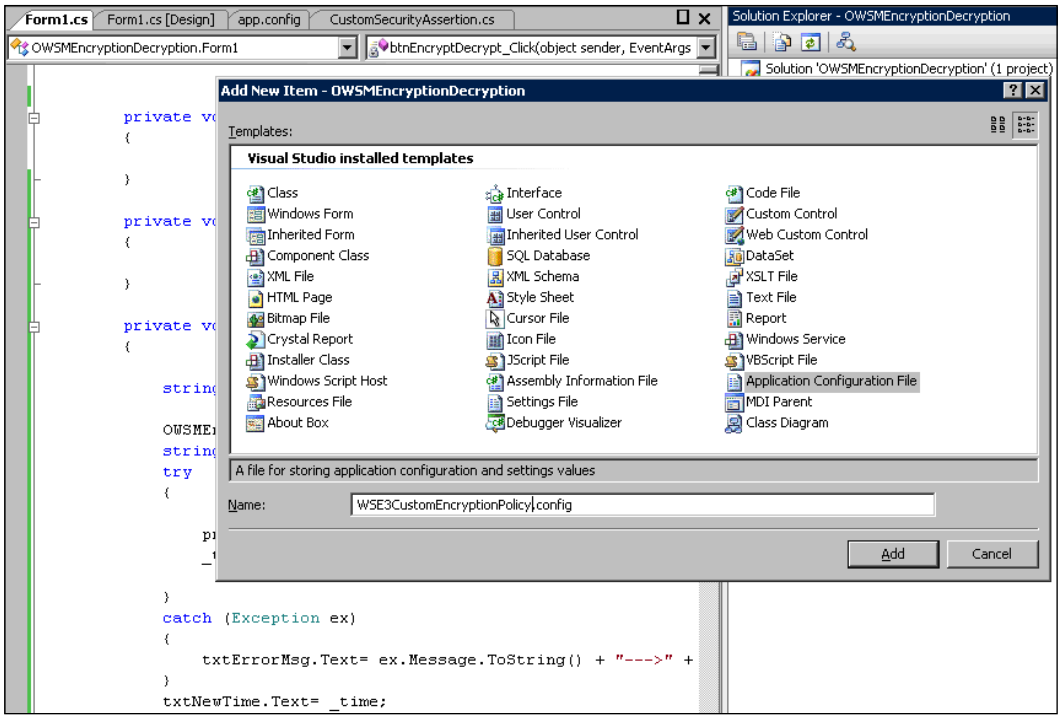
The `CustomSecurityClientInputFilter` is derived from `ReceiveSecurityFilter` and has a method called `ValidateMessageSecurity` which can be overridden to decrypt the message.

The custom security assertion code would look like:

```
          public override SoapFilter CreateClientOutputFilter(
                                FilterCreationContext context)
          {
             // return null;
              //Encrypt outgoing msg
              return new CustomSecurityClientOutputFilter(this);
          }
          public override SoapFilter CreateClientInputFilter(
                               FilterCreationContext context)
          {
  //Decrypt incoming data
              return new CustomSecurityClientInputFilter(this);
          }
```

The `CreateClientOutputFilter` and `CreateClientInputFilter` are overwritten to perform the encryption or decryption. The `SecureMessage` of `CustomSecurityClientOutputFilter` is invoked during the outbound transaction and the `ValidateMessageSecurity` of `CustomSecurityClientIputFIlter` is invoked during the inbound transaction.

Once the custom security is written, it should then be added to the policy. In order to create the policy, you can click on **Add New Item** and then select the configuration file. Then name your policy file as `WSE3CustomEncryptionPolicy.config`, as shown in the next screen capture:



Once the policy file is added, replace the value with the following code:

```
<policies  xmlns="http://schemas.microsoft.com/wse/2005/06/
policy">
  <extensions>
    <extension name="CustomSecurityAssertion" type="OWSMEncryption
Decryption.CustomSecurityAssertion, OWSMEncryptionDecryption" />
  </extensions>
  <policy name="ServicePolicy">
    <CustomSecurityAssertion >
      <clientToken>
        <x509
          storeLocation="CurrentUser"
          storeName="My"
          findValue="CN=WSE2QuickStartClient"
          findType="FindBySubjectDistinguishedName" />
```

```
        </clientToken>
        <serviceToken>
          <x509
            storeLocation="LocalMachine"
            storeName="My"
            findValue="CN=WSE2QuickStartServer"
            findType="FindBySubjectDistinguishedName" />
        </serviceToken>
      </CustomSecurityAssertion >
    </policy>
  </policies>
```

The previous policy information describes the location of the certificate along with a policy name. Once the policy is added, it should be included with the application configuration file so that the policy can be loaded at run time. The `app.config` can be replaced with the following to include the custom assertion extensions and policy extensions.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <configSections>
        <sectionGroup name="applicationSettings" type="System.
        Configuration.ApplicationSettingsGroup, System,
        Version=2.0.0.0, Culture=neutral,
        PublicKeyToken=b77a5c561934e089" >
            <section name="OWSMEncryptionDecryption.Properties.
            Settings" type="System.Configuration.
            ClientSettingsSection, System, Version=2.0.0.0,
            Culture=neutral, PublicKeyToken=b77a5c561934e089"
            requirePermission="false" />
        </sectionGroup>
      <section name="microsoft.web.services3" type="Microsoft.Web.
      Services3.Configuration.WebServicesConfiguration, Microsoft.
      Web.Services3, Version=3.0.0.0, Culture=neutral,
      PublicKeyToken=31bf3856ad364e35" />
    </configSections>
    <applicationSettings>
        <OWSMEncryptionDecryption.Properties.Settings>
            <setting name="OWSMEncryptionDecryption_
                           OWSMEncryptionDecryption_TimeService"
                serializeAs="String">
                <value>http://owsm.packtpub.com:3115/gateway/
                           services/SID0003016</value>
            </setting>
        </OWSMEncryptionDecryption.Properties.Settings>
```

```
    </applicationSettings>
  <microsoft.web.services3>
    <policy fileName="..\..\WSE3CustomEncryptionPolicy.config" />
  </microsoft.web.services3>
</configuration>
```

The policy file has been created, application configuration has been modified, and the custom assertion has been written. We can now write the code for the button click event to invoke the web service and attach the policy, so that the request can be encrypted and response can be decrypted.

The button click code is straightforward. All it does is:

- Gather input from the text boxes
- Instantiate the web service proxy with a WSE enabled extension
- Attach the service policy to the proxy
- Invoke the method

The code will look like:

```
string TextThatWillbeEncrypted = txtInputString.Text;
OWSMEncryptionDecryption.TimeServiceWse proxy =
new global::OWSMEncryptionDecryption.
OWSMEncryptionDecryption.TimeServiceWse();
string _time = "";
try
{
    proxy.SetPolicy("ServicePolicy");
    _time = proxy.getTime(TextThatWillbeEncrypted);
}
catch (Exception ex)
{
    txtErrorMsg.Text= ex.Message.ToString() + "--->" +
    ex.InnerException.Message.ToString();
}
txtNewTime.Text= _time;
```

If the configurations are correct, you will have the new time in the text box when you run the application.

Summary

In this chapter, we learnt how to configure Oracle WSM to decrypt the incoming SOAP message and encrypt the response messages. Though encryption can protect the confidentiality of the message, the security is stronger when combined with a digital signature. In this case, the message is digitally signed first and then encrypted. In the next chapter, we will discuss how to digitally sign and verify the SOAP messages.

6
Digitally Signing and Verifying Messages in Web Services

Confidentiality and integrity are two critical components of web services. In the last chapter, we discussed how support for encryption and decryption in web services addressed the confidentiality issue. While confidentiality can be ensured by means of encryption, the encrypted data can still be overwritten and the integrity of the message can be compromised. It is equally important to protect the integrity of the message, and digital signatures can help protect the integrity of the message. In this chapter, I will describe in detail how to digitally sign and verify messages in web services using Oracle Web Services Manager.

Overview of Digital Signatures

In the web services scenario, XML messages are exchanged between the client application and the web services. Certain messages contain critical business information and, therefore, the integrity of the message should be ensured. Ensuring the integrity of the message is not a new concept, it has been there for a long time. The concept is to make sure that the data was not tampered with in transit between the sender and the receiver.

Consider, for example, that Alice and Bob are exchanging emails that are critical to business. Alice wants to make sure that Bob receives the correct email that she sent and no one else tampered with or modified the email in between. In order to ensure the integrity of the message, Alice digitally signs the message using her private key, and when Bob receives the message, he will check to make sure that the signature is still valid before he can trust or read the email.

What is this digital signature? And how does it prove that no one else tampered with the data? When a message is digitally signed, it basically follows these steps:

- Create a digest value of the message (a unique string value for the message using a SHA1 or MD5 algorithm).
- Encrypt the digest value using the private key, known only to the sender.
- Exchange the message along with the encrypted digest value.

Note: MD5 and SHA1 are message digest algorithms to calculate the digest value. The digest or hash value is nothing but a non-reversible unique string for any given data, i.e. the digest value will change even if a space is added or removed. SHA1 produces a 160 bit digest value, while MD5 produces a 128 bit value.

When Bob receives the message, his first task is to validate the signature. Validation of signature goes through a sequence of steps:

- Create a digest value of the message again using the same algorithm.
- Encrypt the digest value using the public key of Alice (obtained out of band or part of message, etc.)
- Validate to make sure that the digest value encrypted using the public key matches the one that was sent by Alice.
- Since the public key is known or exchanged along with the message, Bob can check the validity of the certificate itself.

Note: Digital certificates are issued by a trusted party such as Verisign. When a certificate is compromised, you can cancel the certificate, which will invalidate the public key.

Once the signature is verified, Bob can trust that the message was not tampered with by anyone else. He can also validate the certificate to make sure that it is not expired or revoked, and also to ensure that no one actually tampered with the private key of Alice.

Digital Signatures in Web Services

In the last section, we learnt about digital signatures. Since web services are all about interoperability, digital-signature-related information is represented in an industry standard format called XML Signature (standardized by W3C). The following are the key data elements that are represented in an interoperable manner by XML Signature:

- What data (what part of SOAP message) is digitally signed?
- What hash algorithm (MD5 or SHA1) is used to create the digest value?
- What signature algorithm is used?
- Information about the certificate or key.

In the next section, we will describe how the Oracle Web Services Manager can help generate and verify signatures in web services.

Signature Generation Using Oracle WSM

Oracle Web Services Manager can centrally manage the security policy, including digital signature generation. One of the greatest advantages in using Oracle WSM to digitally sign messages is that the policy information and the **digital certificate** information are centrally stored and managed.

An organization can have many web services and some of them might exchange certain business critical information and require that the messages be digitally signed. Oracle WSM will play a key role when different web services have different requirements to sign the message or when it is required to take certain actions before or after signing the message. Oracle WSM can be used to configure the signature at each web service level and that reduces the burden of deploying certificates across multiple systems. In this section, we will discuss more about how to digitally sign the response message of the web service using Oracle WSM.

Sign Message Policy Step

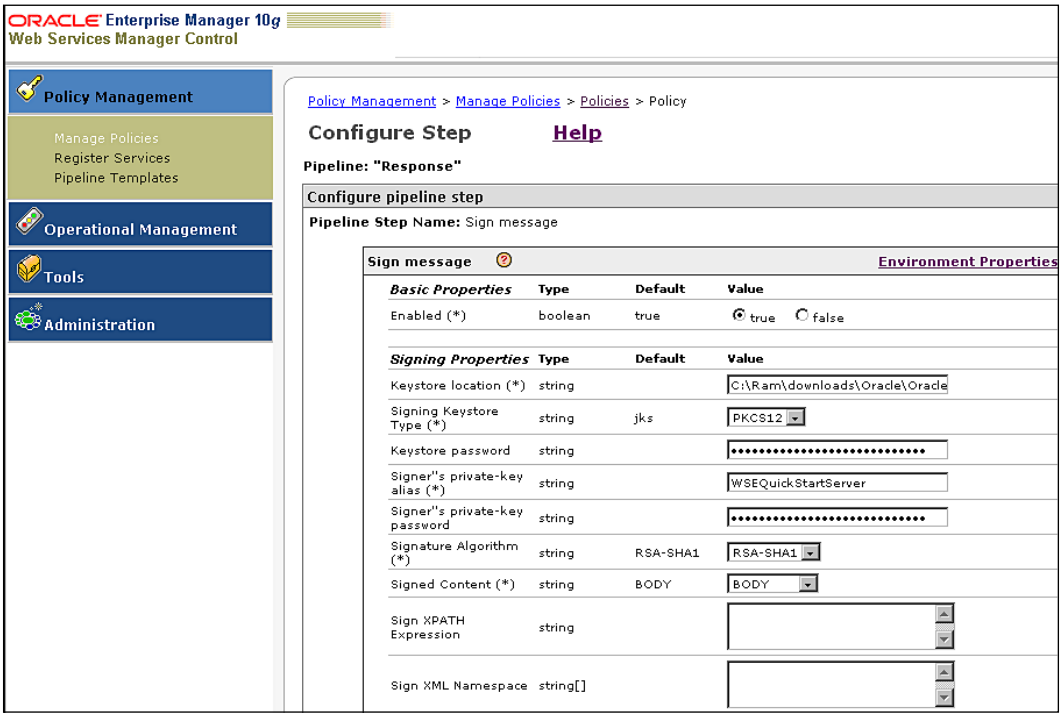
As a quick refresher, in Oracle WSM, each web service is registered within a gateway or an agent and a policy is attached to each web service. The policy steps are divided mainly into request pipeline template and response pipeline template, where different policies can be applied for request or response message processing. In this section, I will describe how to configure the policy for a response pipeline template to digitally sign the response message. (Note: Pipeline templates are reusable templates that can be applied across various policies and are described in detail in Chapter 3)

Note: It is assumed that the web service is registered within a gateway and a detailed example will be described later in this chapter.

In the response pipeline, we can add a policy step called **Sign Message** to digitally sign the message. In order to digitally sign a message, the key components that are required are:

- Private key store
- Private key password
- The part of SOAP message that is being signed
- The signature algorithm being used

The following screenshot describes the "**Sign Message**" policy step with certain values populated.



In the previous screenshot, the values that are populated are:

- **Keystore location**—The location where the private key file is located.
- **Keystore type**—Whether or not it is **PKCS12** or **JKS**.
- **Keystore password**—The password to the keystore.
- **Signer's private-key alias**—The alias to gain access to the private key from the keystore.

- **Signer's private-key password**—The password to access the private key.
- **Signed Content**—Whether the **BODY** or envelope of the SOAP message should be signed.

The above information is a part of a policy that is attached to the time service which will sign the response message. As per the information that is shown in the screenshot, the **BODY** of the SOAP message response will be digitally signed using the **SHA1** as the digest algorithm, and **PKCS12** key store. Once the message is signed, the SOAP message will look like:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope soap:encodingStyle="http://schemas.xmlsoap.org/
soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:
soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:
soapenc="http://schemas.xmlsoap.org/soap/encoding/">
   <soap:Header>
         <wsse:Security xmlns:wsse="http://docs.oasis-open.
         org/wss/2004/01/oasis-200401-wss-wssecurity-secext-
         1.0.xsd" xmlns="http://docs.oasis-open.org/
         wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
         soap:mustUnderstand="1">
               <wsse:BinarySecurityToken ValueType="http://docs.
               oasis-open.org/wss/2004/01/oasis-200401-wss-
               x509-token-profile-1.0#X509v3"
               EncodingType="http://docs.oasis-open.
               org/wss/2004/01/oasis-200401-wss-soap-message-
               security-1.0#Base64Binary" wsu:Id="_
               VLL9yEsi09I9f5ihwae2lQ22" xmlns:wsu="http://docs.
               oasis-open.org/wss/2004/01/oasis-200401-wss-
               wssecurity-utility-1.0.xsd">SecurityTOkenoKE2ZA==<
               /wsse:BinarySecurityToken>
               <dsig:Signature xmlns="http://www.w3.org/2000/09/
               xmldsig#" xmlns:dsig="http://www.w3.org/2000/09/
               xmldsig#">
                     <dsig:SignedInfo>
                           <dsig:CanonicalizationMethod
                           Algorithm="http://www.w3.org/2001/10/
                           xml-exc-c14n#"/>
                           <dsig:SignatureMethod
                           Algorithm="http://www.w3.org/2000/09/
                           xmldsig#rsa-sha1"/>
                           <dsig:Reference URI="#ishUwYWW2AAthrx
                           hlpv1CA22">
                                 <dsig:Transforms>
                                       <dsig:Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
```

```
                                    </dsig:Transforms>
                                    <dsig:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                                    <dsig:DigestValue>ynuqANuYM3qzh
dTnGOLT7SMxWHY=</dsig:DigestValue>
                              </dsig:Reference>
                              <dsig:Reference URI="#UljvWiL8yjedImz
                               6zy0pHQ22">
                                    <dsig:Transforms>
                                          <dsig:Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                                    </dsig:Transforms>
                                    <dsig:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                                    <dsig:DigestValue>9ZebvrbVYLiPZ
v1BaVLDaLJVhwo=</dsig:DigestValue>
                              </dsig:Reference>
                        </dsig:SignedInfo>
                        <dsig:SignatureValue>QqmUUZDLNeLpAEFXndiBLk=
</dsig:SignatureValue>
                        <dsig:KeyInfo>
                              <wsse:SecurityTokenReference
xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd" wsu:Id="_7vjdWs1ABULkiLeE7Y4lAg22"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-wssecurity-utility-1.0.xsd">
                                    <wsse:Reference URI="#_
VLL9yEsi09I9f5ihwae2lQ22"/>
                              </wsse:SecurityTokenReference>
                        </dsig:KeyInfo>
                  </dsig:Signature>
                  <wsu:Timestamp xmlns:wsu="http://docs.oasis-open.
org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd" wsu:Id="UljvWiL8yjedImz6zy0pHQ22">
                        <wsu:Created>2007-11-16T15:13:48Z</wsu:
Created>
                  </wsu:Timestamp>
            </wsse:Security>
      </soap:Header>
      <soap:Body wsu:Id="ishUwYWW2AAthrxhlpv1CA22" xmlns:wsu="http://
docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd">
            <n:getTimeResponse xmlns:n="urn:Test:GetTime">
                  <Result xsi:type="xsd:string">10:13 AM</Result>
            </n:getTimeResponse>
      </soap:Body>
</soap:Envelope>
```

Internals of Sign Message Policy Step

The example XML message is nothing but the SOAP response message that is digitally signed by Oracle WSM. The parameters that are selected or that can be selected will affect the message output and hence it is important to understand how Oracle WSM assembles the digitally signed message.

First of all, Oracle WSM was configured to sign the SOAP body message, which in our example is the actual time from the server. In the above example, the SOAP body is referenced by the identifier `ishUwYWW2AAthrxhlpv1CA22`. Only the SOAP body message should be digitally signed. We understand that in the signature generation process, we should first calculate the digest value of the message and then encrypt the digest value. In the sample XML, there are a few components which are worth explaining.

Reference Element

The `Reference` element describes what part of the message is hashed and what digest algorithm is used to create the hash value and any transformations applied before the digest was calculated.

Let's consider a portion of our signed response message and in the message below, we will notice that the `DigestMethod` is `SHA1` and, that the `DigestValue` is also embedded.

```
<dsig:Reference URI="#ishUwYWW2AAthrxhlpv1CA22">
      <dsig:Transforms>
            <dsig:Transform Algorithm="http://
           www.w3.org/2001/10/xml-exc-c14n#"/>
      </dsig:Transforms>
      <dsig:DigestMethod Algorithm="http://www.
      w3.org/2000/09/xmldsig#sha1"/>
      <dsig:DigestValue>ynuqANuYM3qzhdTnGOLT7SMxWH
      Y=</dsig:DigestValue>
</dsig:Reference>
```

In the above example, there is also an element called `Transforms` which contains a list of transform elements. The `Transform` element describes what transformation is applied to the XML message before the digest is calculated. In our example, the **Exclusive Canonicalization** transformation is used.

Note: Since the digest values can differ even when a space is added or removed, canonicalization transformation will transform the data to an accepted format before the digest is calculated.

SignedInfo Element

The `SignedInfo` element describes the actual signature algorithm and a list of references that contain the digest value of the message. In our case, the signature algorithm is `rsa-sha1` and it also contains a reference to the SOAP `Body` element (`Id` attribute of the SOAP `Body` element). In our example of signed SOAP response message, there are actually two reference elements: one that refers to the SOAP `Body` and an other that refers to the `Timestamp` element. The `SignatureMethod` element describes the actual signature algorithm used.

Signature

The `Signature` element is the root element that describes the digital signature. It contains the `SignedInfo` element, `SignatureValue` and the `KeyInfo` element. The `SignatureValue` element contains the actual signature value and the `KeyInfo` element contains information about the certificate.

Signature Generation and Verification Example

In an earlier section, we learnt how the sign message policy step can be configured to digitally sign the message and also the internals of how Oracle WSM creates the signed SOAP response message. In this section, we will walk through the signature generation and verification process within Oracle WSM by means of an example.

In this example, we will have the same time web service which will be registered within Oracle WSM Gateway. Oracle WSM will validate the incoming signed SOAP message and then will respond with a signed SOAP message. In this example we will demonstrate how:

- To register web service with Oracle WSM
- Oracle WSM can be configured to validate the signature in SOAP request
- A Microsoft .NET application can digitally sign the SOAP request
- Oracle WSM can be configured to sign the response SOAP message
- A Microsoft .NET application can validate the signature of the SOAP message

Registering Web Service with Oracle WSM

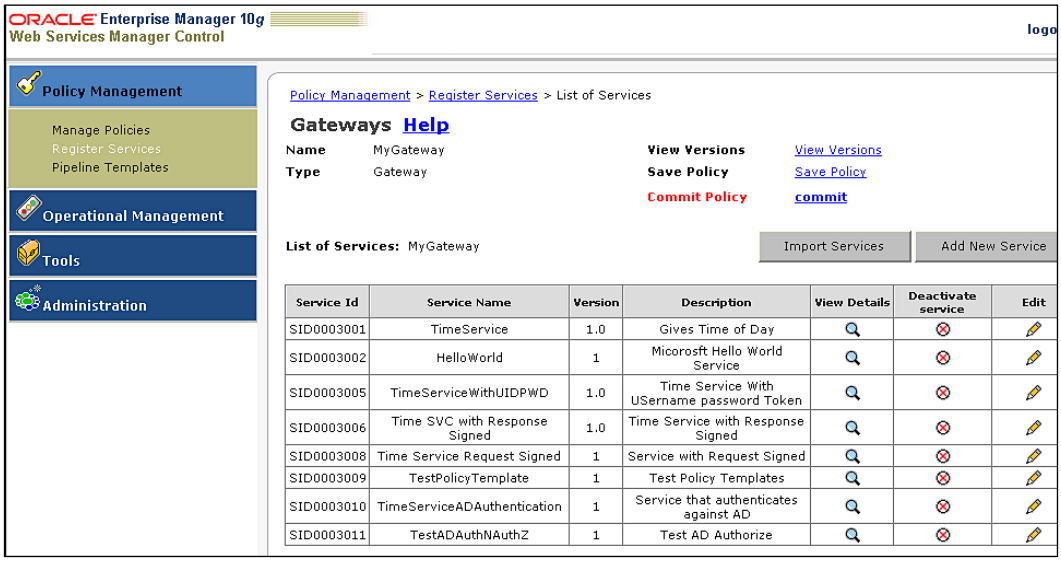
In order to protect a web service within Oracle WSM, the first step is to register the web service within Oracle WSM and then edit the policy associated with it. The following steps describe how to register the service:

Login to Oracle Web Service Manager Console at:

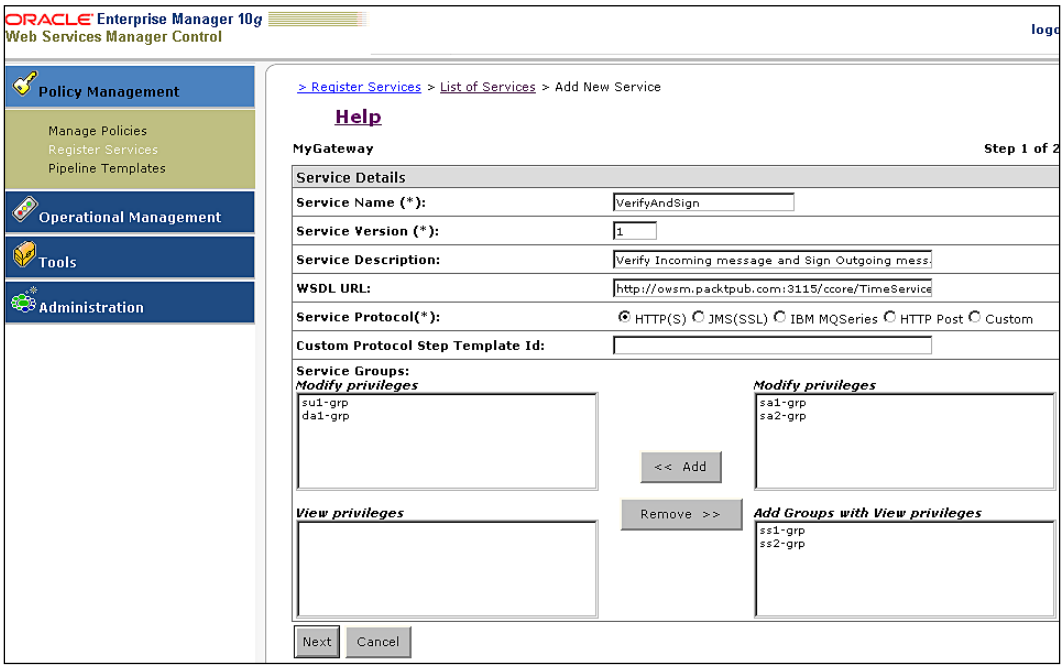
```
http://owsm.packtpub.com:3115/ccore
```

Click on **Policy Management** and then **Manage Services**; you will see the **List of Gateways** that are available as shown in the following screenshot.

On the right side of the screen, if you click on the **Services** hyperlink, you will see the list of registered services (refer to the following screenshot).



In order to add a new service, click on **Add New Service** on the right side panel.



The previous screenshot show the details that can be added while adding the new service. The * after each label makes those fields mandatory. The screen asks for typical information such as:

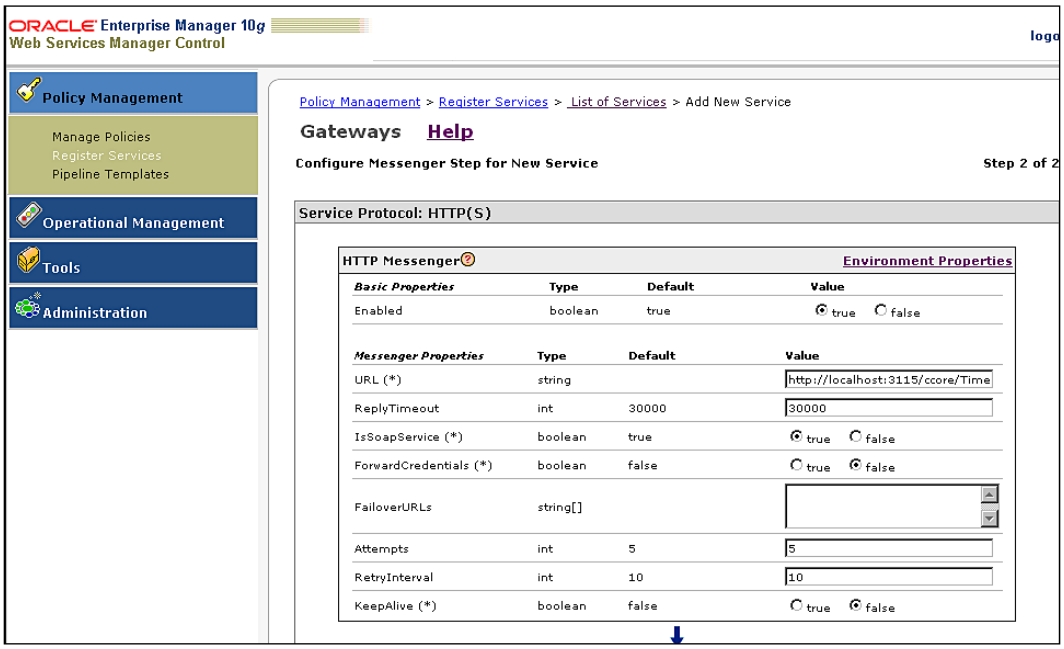
- Name of the service
- Version of the service
- Any description of the service
- **WSDL URL** of the service
- Protocol in which service will accept messages

It also asks for additional information such as **Service Groups**, groups that are part of Oracle WSM with the right to view and with the right to update.

In our example, we are registering time service that will validate the signature of the web service request and then will sign the response message. The time service is registered with the following information:

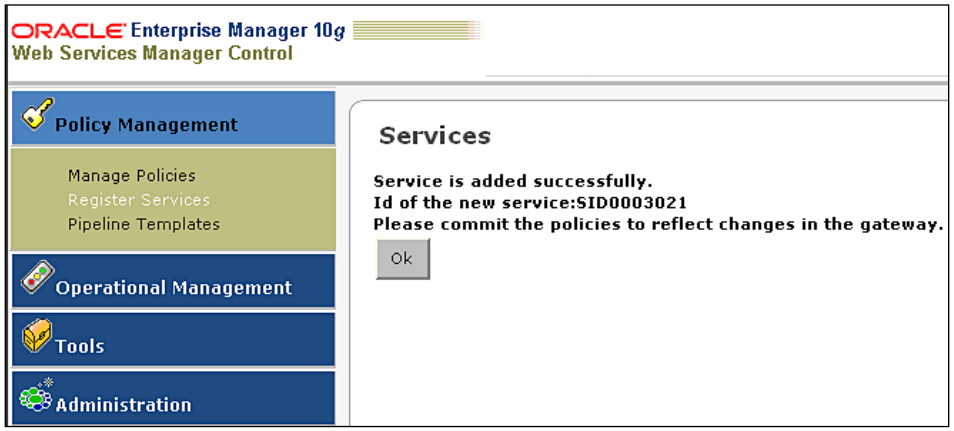
- Service Name: VerifyAndSign
- Service Version: 1
- Service Description: Verify Incoming message and Sign Outgoing message
- WSDL URL: `http://owsm.packtpub.com:3115/ccore/TimeService.wsdl`
- Service Protocol: HTTP(S)
- Service Groups: Select the default that has full permissions

Once the information is filled out, click **Next** on **New Service** registration. This will take you to the next screen which will display the actual **URL** of the service (refer to the following screenshot).



The URL in this page comes from the WSDL URL. You just have to make sure that the service is enabled and check if it is SOAP service or not.

You can then click **Finish** to register the service. Once you click **Finish**, the Oracle WSM internally generates a new service ID and now the client applications can use that service ID to communicate.



The previous screenshot show that the Oracle WSM registered the time service and created a new service ID.

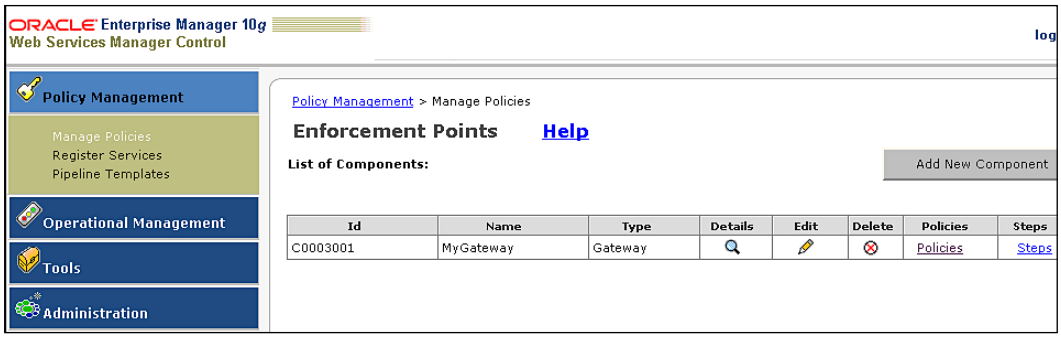
Click **OK** to get back to the main screen that lists all the services.

We have only added the new service, but it hasn't been committed yet. You can now click **OK** to commit the policy.

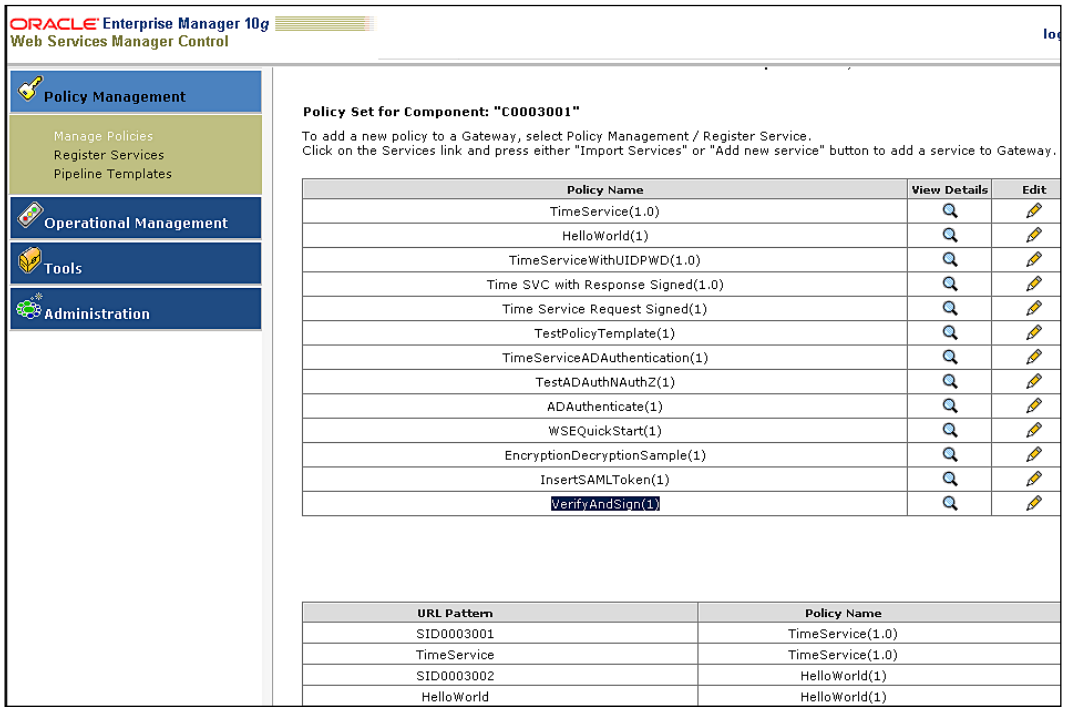
Signature Verification by Oracle WSM

Oracle Web Services Manager can actually validate the signature in the incoming i.e. request SOAP message. By using Oracle WSM to validate the signature, organizations can actually centralize the policy enforcement and also the public key management. As organizations deploy more web services that are accessed by other divisions and business partners, managing the signature verification process might become tedious, as with each new consumer, the certificate information should be maintained. Oracle WSM can address such issues by centralizing those operations. In this section I will describe how to configure Oracle WSM policy to validate the signature of the SOAP request message.

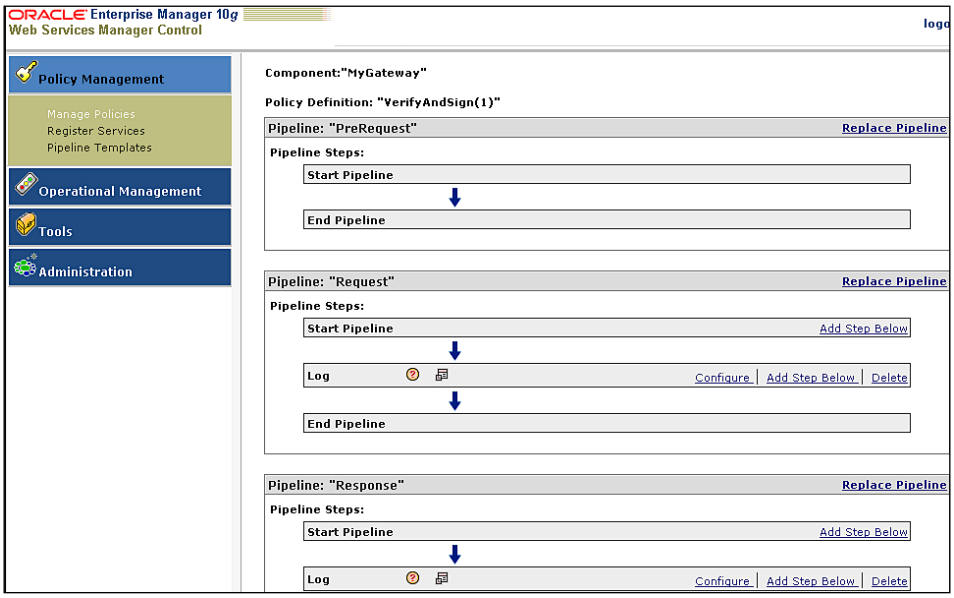
In order to view the policy, you can click on **Policy Management** and then **Manage Policies**. This will bring you to the screen with the gateway information and a hyperlink for policies (see the following screen capture).



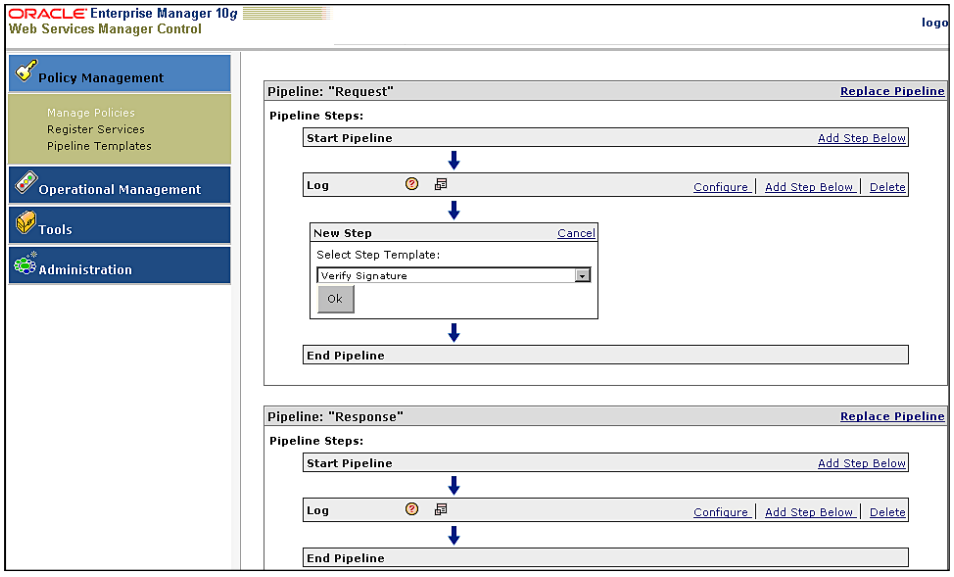
You can then click on **Policies** to see all the policies and you will see the **VerifyAndSign** policy too that is created by default.



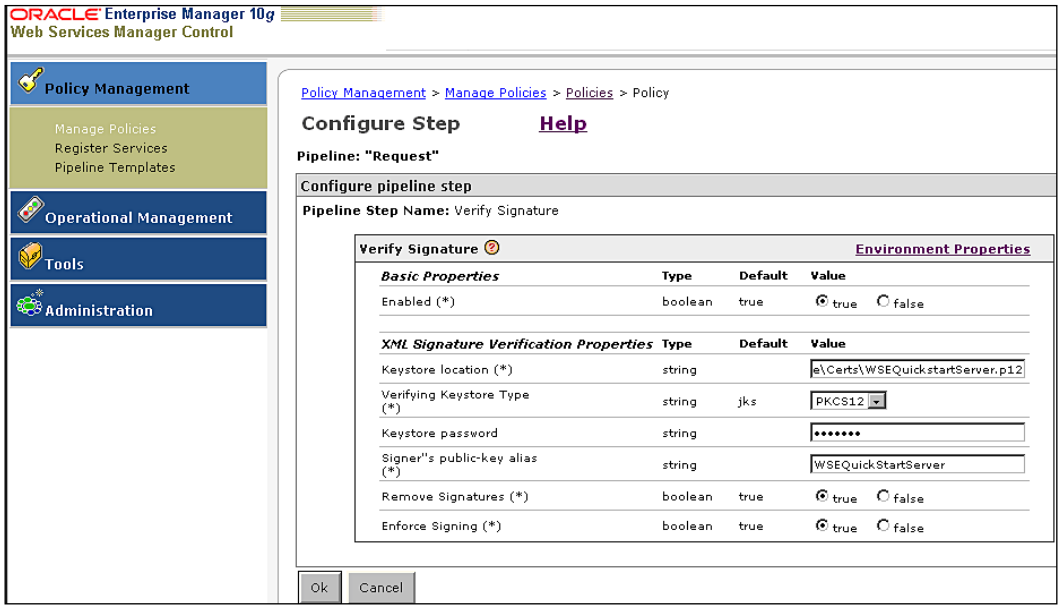
A default policy is attached to the service. We can now click **Edit** to edit the policy. When you click **Edit**, you will see the policy steps as shown in the following screenshot.



In this section, we want to configure the **Request** pipeline to validate the signature of the incoming SOAP message. In order to validate the signature, click **Add Step Below** to add the **Verify Signature** policy step as shown in the following screenshot.



Once you click **OK**, the verify signature policy step is added, but that policy step should be configured. If you click on the **Configure** button on the verify signature policy step, it will take you to the screen where you can configure the verify signature policy information as shown in the following screen capture.



In the previous screenshot, I configured **Verify Signature** policy steps with:

- Location of the key store
- Key store type as **PKCS12**
- Password of the key store
- Public key alias in the key store
- Set **Remove Signatures** to **true** to remove the digital signature after the signature validation
- **Enforce Signing** is set to **true** to make sure that the incoming requests are signed

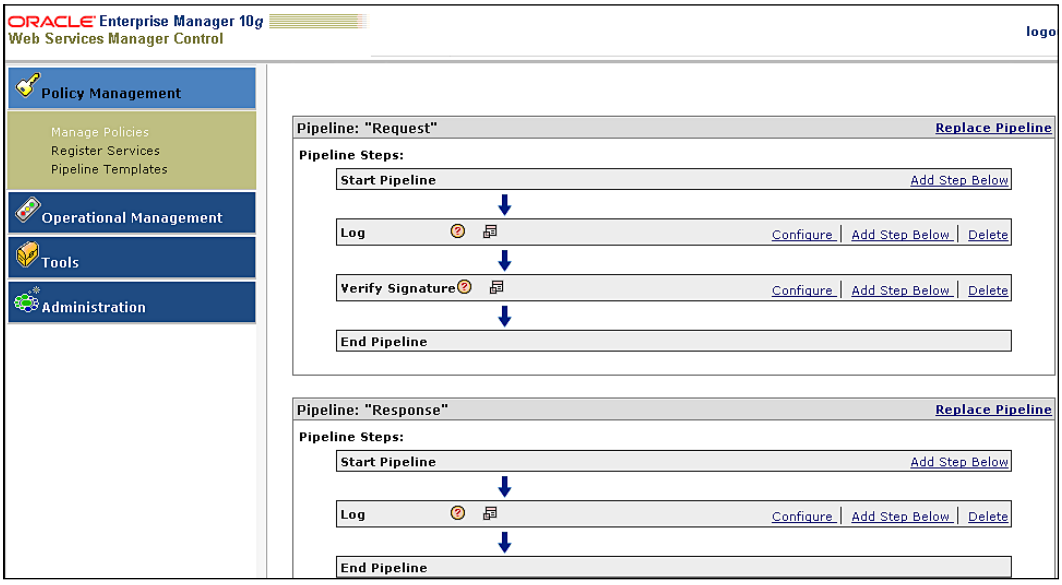
Note: In order to generate a PKCS12 key store from certificate that is installed already in Microsoft certificate services, you should first export the certificate (with or without private key) and then import that certificate in FireFox (Advanced option) and then export back to PKCS12.

Once the verify signature policy has been configured and saved (Commit Policy), the policy would enforce that any request for the time service with the particular service ID be digitally signed.

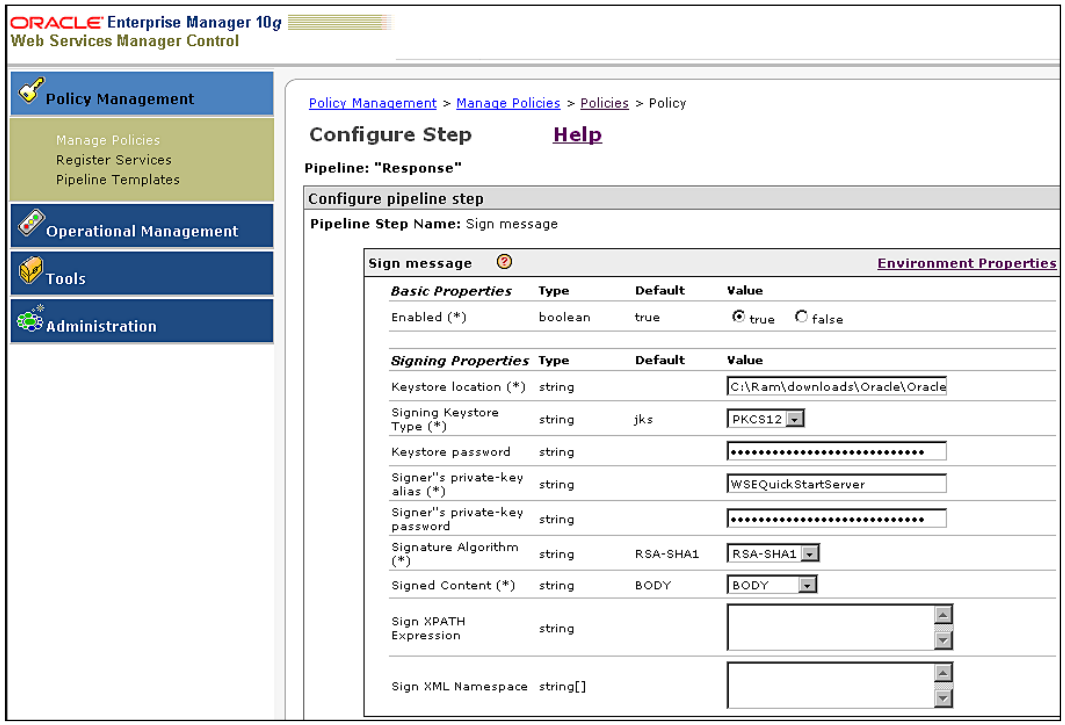
Signature Generation by Oracle WSM

In the last section, we discussed how to digitally sign a web service request by Microsoft .NET application and how to validate the signature by Oracle WSM. In this section, we will discuss how to digitally sign the web service response message. In the earlier section, we discussed how to register the service and how to attach the verify signature policy step to the request pipeline.

In order to digitally sign the response message, the response pipeline of the policy should be modified to include the sign message policy step. The policy with the request pipeline that is already configured to verify signature would look like:

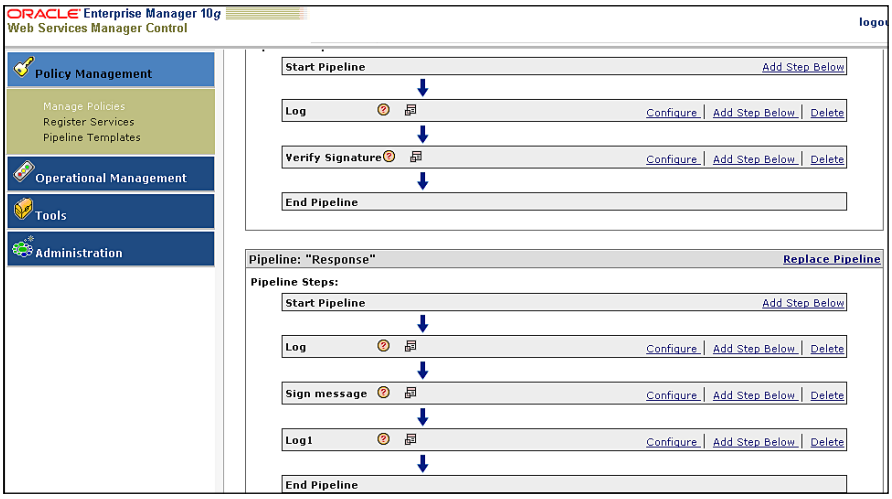


Now we have to add the step in the **Response** pipeline to actually sign the response message. In order to add the policy step, click on **Add Step Below** and then select the **Sign Message** policy step. Once the Sign Message policy step is added, it can then be configured, as shown in the following screenshot, to include the appropriate key store location for the public key to digitally sign the message.



In the previous figure, the location of the key store that has the private key, along with the **Keystore password**, alias and part of message to be signed are specified.

Once the policy is created, it would look like:



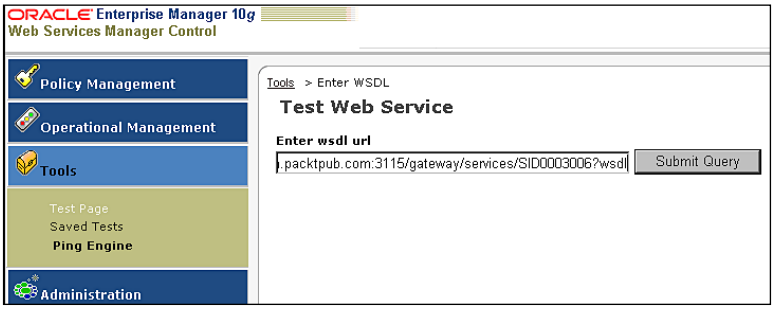
In the previous screenshot, the **Response** pipeline has two log steps—one to log the message before digitally signing and one to log the message after digitally signing the message. In this sample, we are using the same WSEQuickStartServer certificate to sign the message.

Once the policy is saved, the response message will be digitally signed. The client application (Microsoft .NET) can be configured to validate the signature.

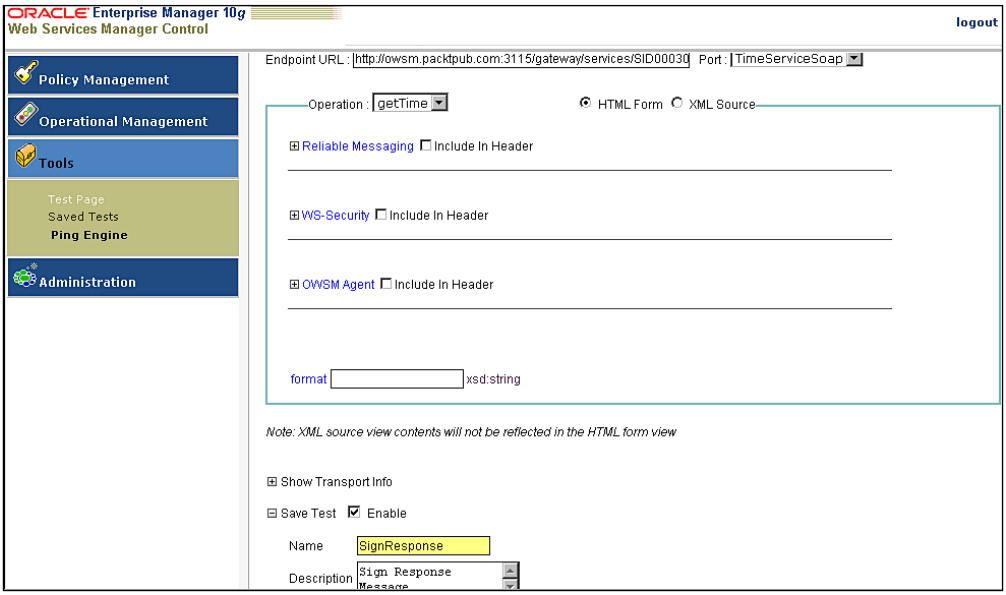
Oracle WSM Test Page as Client Application

Oracle WSM comes with its own test page where you can test the web service and the security policy associated with the web service. In this example, we will show how to test the web service policy that was just deployed and which digitally signs the response message.

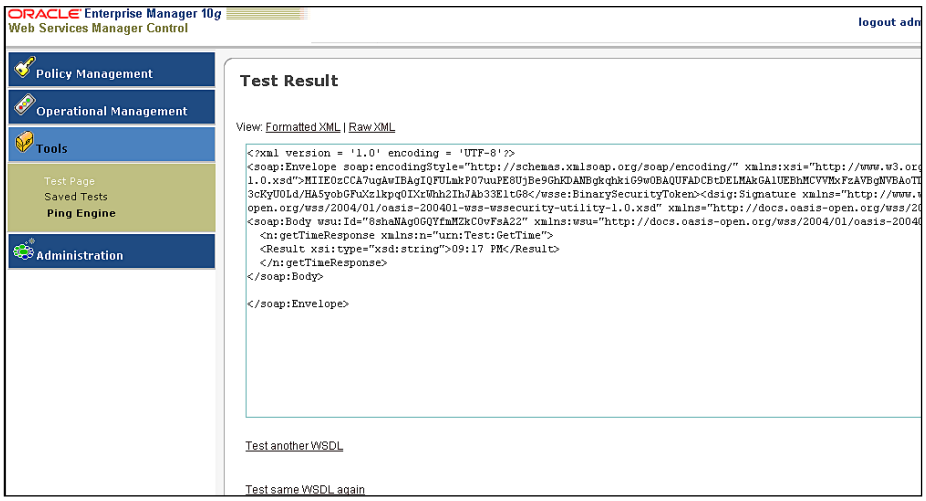
You can get to the test page from the **Tools** menu.



In the WSDL URL text box, enter the WSDL URL and then click on **Submit Query**. It will come up with a window to enter any credentials (username and password) and specify if that should be sent in the HTTP header or as a part of the SOAP message. It also has an option to save the test as shown in the following screen capture.



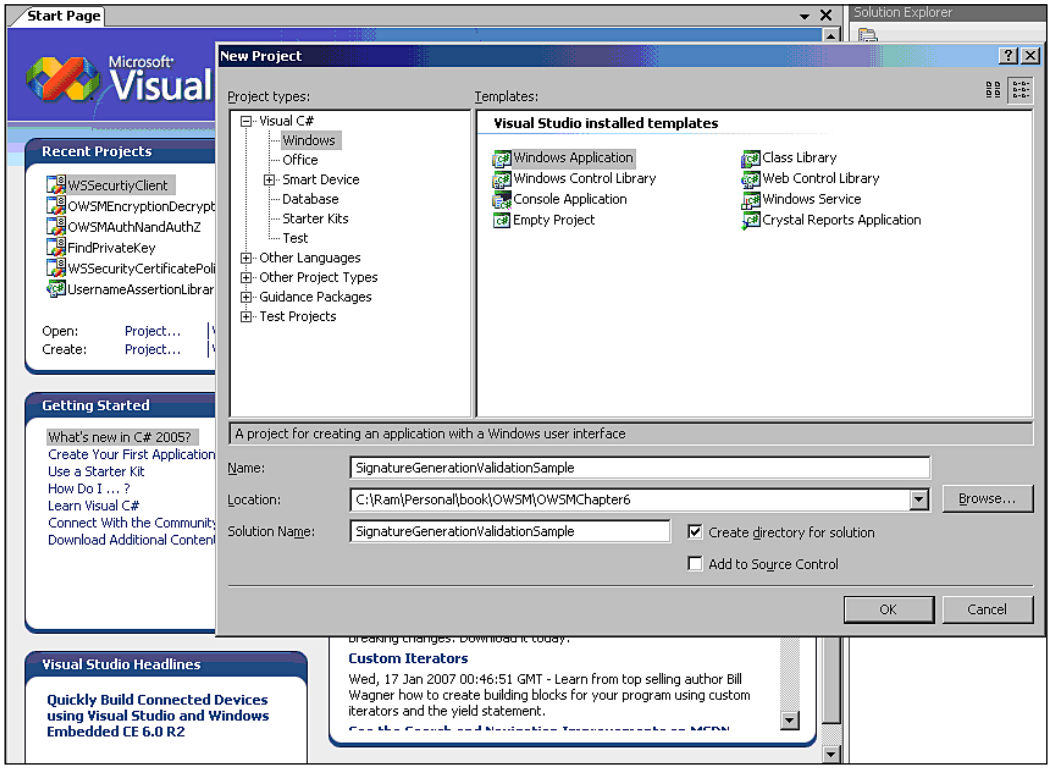
You can give a name for the test and any description and then click **Invoke**. When you click the **Invoke** button, the web service is invoked and the test is also saved. In our example, once the web service is invoked, the security policy is applied and the response message is digitally signed as shown in the next screenshot.



In the next example, you will see how to create a client application in Microsoft .NET to perform the signature generation and validation.

Microsoft .NET Client Application

The client application is a Windows Forms application written using Microsoft Visual Studio 2005 with Microsoft WSE 3.0. The first step in creating a client application is to create a Windows Forms application called **SignatureGenerationValidationSample**. The following is a screenshot on how to create the Windows Forms application (as described in last chapter).



Once you click **OK**, the Windows Forms application is created with a default form.

Since we are going to invoke the time web service with input data, we will add a text box to enter any random input data and a button to invoke the web service. We will also add another text box to show the **Error** message if any (refer to the following screenshot).

Now we should add the Microsoft WSE 3.0 as a reference before we can create the proxy class. You can add the WSE reference by selecting **Add Reference** from the project and then selecting Microsoft.Web.Services3.

Now that WSE3 is added, we can add a web reference to the web service to create the proxy object. It can be added by right-clicking on the **Add Web Reference** and then entering the WSDL URL and clicking **OK** to generate the proxy.

So far we have:

- Created the Windows Forms application
- Added a reference to WSE3.0
- Created the web service proxy

The next step is to configure and write code to digitally sign the message. Though it is beyond the scope of this book to explain in detail how to configure policy and write custom policy assertions, we will take a high level overview of how Microsoft .NET and WSE 3.0 can be used to sign and verify SOAP messages.

With Microsoft .NET 3.0 and WSE 3.0, the web services security can be applied either through:

- Policy configuration for certain frequently-used security configurations
- Custom assertion in combination with policy to perform all other security operations

Since our example focuses only on signing the request and validating the response, we will have to write a custom policy assertion so that SOAP messages can be interpreted both during request and response cycles to perform signature generation and validation. The next step is to write the custom assertion that will:

- Sign the outgoing message
- Verify the incoming message

At a high level, the custom assertion inherits from `SecurityPolicyAssertion` class and has methods such as `CreateClientOutputFilter` and `CreateClientInputFilter` that can be overridden to perform any operation at the client side. In our sample, we have created two new classes called `CustomSecurityClientInputFilter` and `CustomSecurityClientOutputFilter` to perform any data processing on request or response.

The `CustomSecurityClientOutputFilter` is derived from `SendSecurityFilter` and has a method called `SecureMessage` which can be overridden to digitally sign the message.

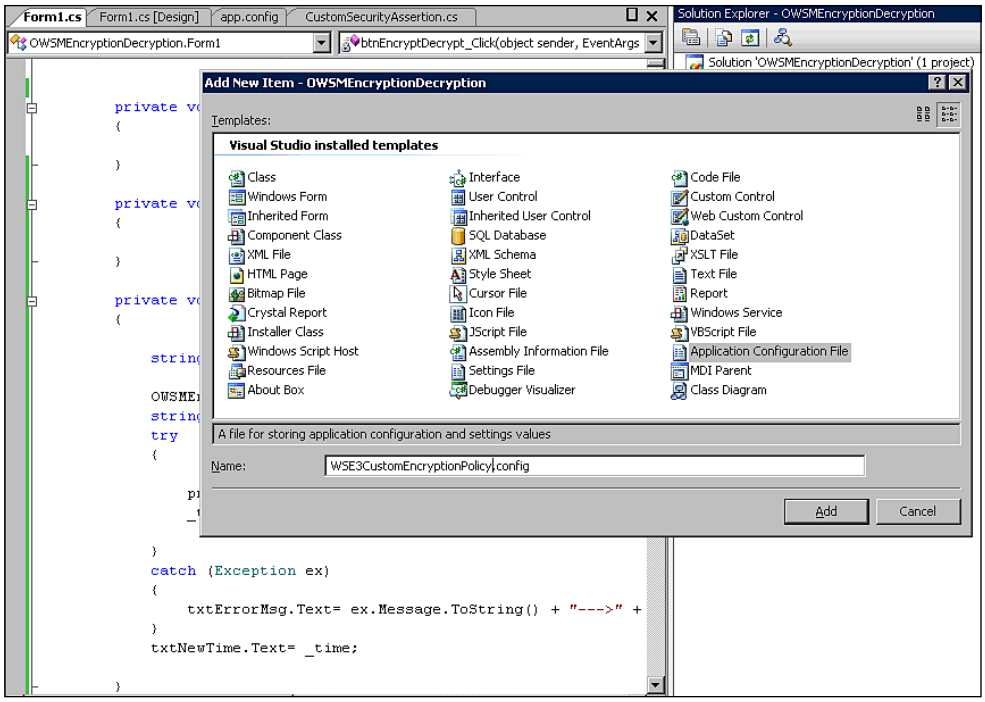
The `CustomSecurityClientInputFilter` is derived from `ReceiveSecurityFilter` and has a method called `ValidateMessageSecurity` which can be overridden to validate the signature of the message.

The custom security assertion code would look like:

```
          public override SoapFilter CreateClientOutputFilter(
                                  FilterCreationContext context)
          {
             // return null;
              //Encrypt outgoing msg
              return new CustomSecurityClientOutputFilter(this);
          }
          public override SoapFilter CreateClientInputFilter(
                                FilterCreationContext context)
          {
  //Decrypt incoming data
              return new CustomSecurityClientInputFilter(this);
          }
```

The `CreateClientOutputFilter` and `CreateClientInputFilter` are overwritten to perform the signature generation or validation. The `SecureMessage` of `CustomSecurityClientOutputFilter` is invoked during the outbound transaction and the `ValidateMessageSecurity` of `CustomSecurityClientIputFIlter` is invoked during the inbound transaction.

Once the custom security is written, it should then be added to the policy. In order to create the policy, you can click on **Add New Item** and then select the configuration file, then name your policy file as `WSE3CustomSignaturePolicy.config` as shown in the following screen capture:



Once the policy file is added, replace the value with the following code:

```
<policies  xmlns="http://schemas.microsoft.com/wse/2005/06/
policy">
  <extensions>
    <extension name="CustomSecurityAssertion" type="Sign
atureGenerationValidationSample.CustomSecurityAssertion,
SignatureGenerationValidationSample" />
  </extensions>
  <policy name="ServicePolicy">
    <CustomSecurityAssertion >
      <clientToken>
        <x509
          storeLocation="CurrentUser"
          storeName="My"
          findValue="CN=WSE2QuickStartClient"
          findType="FindBySubjectDistinguishedName" />
      </clientToken>
      <serviceToken>
        <x509
          storeLocation="LocalMachine"
          storeName="My"
```

```
            findValue="CN=WSE2QuickStartServer"
            findType="FindBySubjectDistinguishedName" />
        </serviceToken>
      </CustomSecurityAssertion >
    </policy>
  </policies>
```

The policy information above describes the location of the certificate along with a policy name. Once the policy is added, it should be included with the application configuration file so that the policy can be loaded at run time. The `app.config` can be replaced with the following to include the custom assertion extensions and the policy extensions.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <configSections>
        <sectionGroup name="applicationSettings" type="System.
Configuration.ApplicationSettingsGroup, System, Version=2.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089" >
          <section name="WSSecurtiyClient.Properties.Settings"
type="System.Configuration.ClientSettingsSection, System,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
"
requirePermission="false" />
          <section name="SignatureGenerationValidationS
ample.Properties.Settings" type="System.Configuration.
ClientSettingsSection, System, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089" requirePermission="false" />
        </sectionGroup>
      <section name="microsoft.web.services3" type="Microsoft.Web.
Services3.Configuration.WebServicesConfiguration, Microsoft.Web.
Services3, Version=3.0.0.0, Culture=neutral, PublicKeyToken=31bf38
56ad364e35" />
    </configSections>
    <applicationSettings>
        <SignatureGenerationValidationSample.Properties.Settings>
            <setting name="SignatureGenerationValidationSample_
SignAndVerify_TimeService"
                serializeAs="String">
                <value>http://cpd10cisslaksh.coxinc.com:3115/
gateway/services/SID0003021</value>
            </setting>
        </SignatureGenerationValidationSample.Properties.Settings>
    </applicationSettings>
```

```
  <microsoft.web.services3>
    <policy fileName="..\..\WSE3CustomSignaturePolicy.config" />
  </microsoft.web.services3>
</configuration>
```

Now that the policy file is created, application configuration is modified, and custom assertion is written, we can write the code for the button click event to invoke the web service and attach the policy so that the request can be encrypted and response can be decrypted.

The button click code is straightforward. All it does is:

- Gather input from the text boxes
- Instantiate the web service proxy with a WSE enabled extension
- Attach the service policy to the proxy
- Invoke the method

The code will look like:

```
            string _Inputdata = txtInput.Text;
            string _Newtime="";
            string _error = "";
            try
            {
                SignAndVerify.TimeServiceWse _Proxy =
new SignatureGenerationValidationSample.SignAndVerify.
TimeServiceWse();
                _Proxy.SetPolicy("ServicePolicy");
                _Newtime = _Proxy.getTime(_Inputdata);
            }
            catch (Exception ex)
            {
                txtError.Text = "";
                _error = ex.Message.ToString();
            }
            txtNewTime.Text = _Newtime;
            txtError.Text = _error;
```

If the configurations are correct, you will find the new time in the text box when you run the application.

Summary

In this chapter, we discussed the importance of digital signatures and how Oracle WSM can be leveraged to digitally sign and verify the messages to ensure the integrity of the message. With digital signatures, especially in validating signatures, each application has to maintain the public key information from each business partner and that will eventually become a tedious task by itself. Oracle WSM can help address such issues by centrally managing the keys and the related policy. In the next few chapters, we will explore how Oracle WSM will be deployed in a real world scenario and how Oracle WSM can help address the security issues and, at the same time, how it is highly available and scalable.

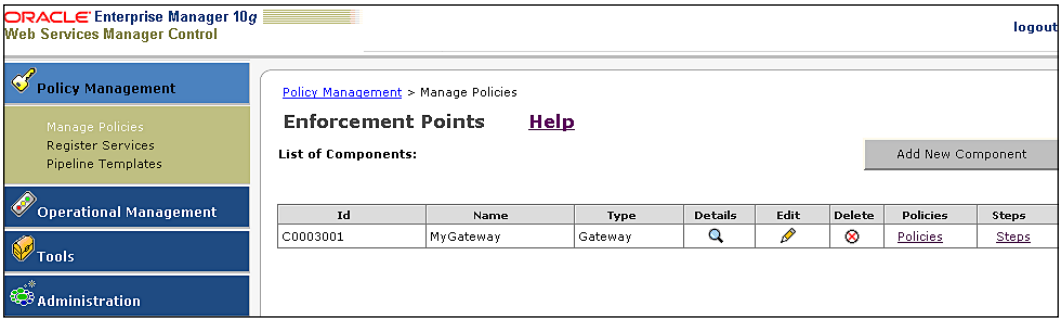
7
Oracle WSM Custom Policy Step

Organizations can externalize their web services security policy definition and enforcement to Oracle Web Services Manager so that the web services' service developers can concentrate on adding business functionality to the services. While Oracle WSM can help define and enforce WS-security policies, there are times where you need to perform additional processing that is not supported by Oracle WSM out of the box, such as validating the IP address of the client invoking the web service or validating custom authentication tokens generated by another identity management product. Under those circumstances, you can implement a custom policy step to address those challenges. Instead of performing those customizations at individual web service provider or consumer level, one can add a custom policy step to Oracle WSM and still derive the benefits of centralizing all the security operations and externalizing security from the application code. In this chapter, we will take a look at how to implement a custom policy step in Oracle WSM.

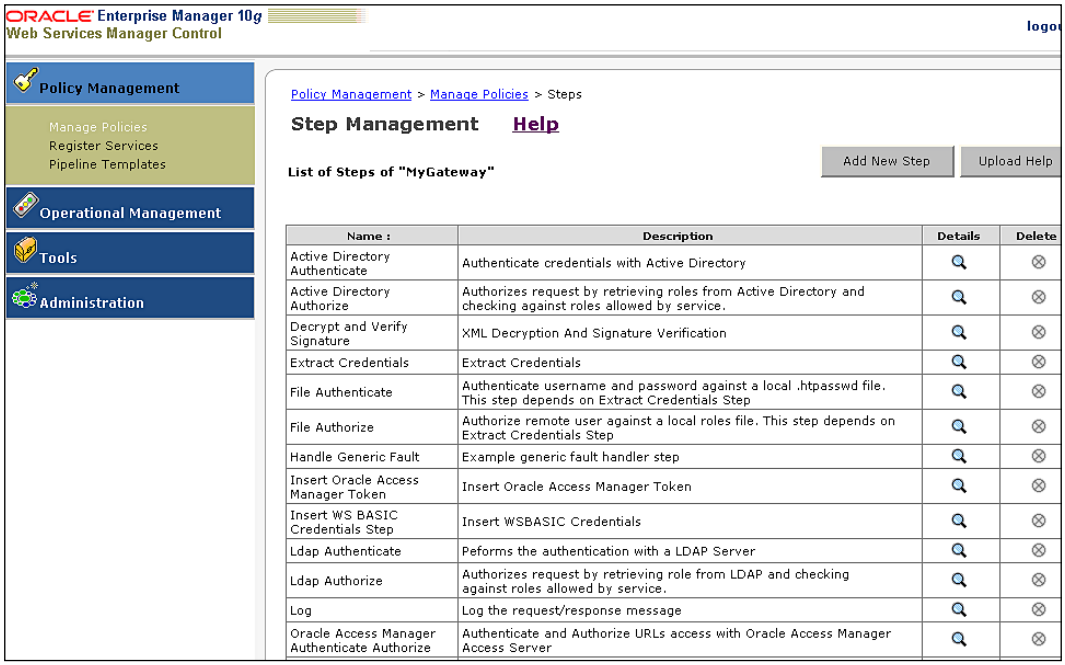
Overview of Oracle WSM Policy Steps

The web services security policies that can be defined within Oracle WSM are a collection of various policy steps. Policy steps are nothing but Java code that extends the `AbstractStep` class and has its own implementation. The policy step that performs active directory authentication has its own Java class that extends the `AbstractStep` with the implementation to authenticate the username and password against the active directory.

In Oracle WSM, you can see the list of policy steps that are available when you click on the **Steps** link from the **Manage Policies** tab on the left side. The following screenshot shows the main screen when you click on **Manage Policies**.



When you click on the **Steps** link, you will see the list of all the policy steps that are available by default with Oracle WSM (refer to the following screenshot).



When defining a security policy for a web service, you will select one or more of these policy steps to create a policy. For instance, if you want to authenticate users against an active directory and then verify a signature, you will use the following policy steps:

- **Extract Credentials**
- **Active Directory Authenticate**
- Verify Signature

In the example above, each policy step performs a set of actions. The **Extract Credentials** will extract the username and password from the SOAP header, the **Active Directory Authenticate** will authenticate the user against active directory and the Verify Signature will validate the incoming digital signature.

While Oracle WSM is shipped with a set of known policy steps, you may have requirements that are not met by the existing policy steps, such as authenticating the username and password that were sent as a part of **UserNameToken**, or authenticating based on the information available in the HTTP header variables, or you may want to modify the request or response messages, etc. When Oracle WSM policy steps don't meet your requirements, you can always implement your own custom policy step that can still be added to list of the steps and included within a policy to secure the web services. In the next section, we will take a closer look at what it takes to implement a custom policy step.

Implementing a Custom Policy Step

Oracle WSM policy step comprises the following components:

- Java class that extends the `AbstractStep` class and implements the `execute` method
- Step template XML file with unique identifier, name and other parameters that are required for the step
- Deployment of custom policy step

The Java class that extends the `AbstractStep` class contains the actual code to execute when the custom policy step is invoked. However, if you want to provide any inputs while adding that policy step to the policy, that information is represented in the step template so that the administrator can add the appropriate values while configuring the policy step. Once the custom policy step implementation is ready and compiled, it needs to be deployed within Oracle WSM and the step template will be used to add the new step. In the next section, we will discuss extending the `AbstractStep` class to develop the custom policy step.

Extending the AbstractStep Class

In order to implement the custom step, you have to implement the `execute` method which is a part of the `com.cfluent.policysteps.sdk.AbstractStep` class. You can also override the `Init` method to add any specific initialization code for the custom policy step, and you can override the `destroy` method to perform any clean up.

When you actually implement the `execute` method, you have to keep in mind the following characteristics of the execute method:

- The `execute` method is the entry point for the custom policy step.
- The `execute` method returns an object of type `IResult`.
- The `execute` method can throw an exception of type `Fault` (`com.cfluent.policysteps.sdk.Fault`).

The sample code below shows a simple custom policy step that extends the `AbstractStep` and has implemented the `execute` method. The `execute` method implementation in this example is nothing but creating the `Result` object and setting the `Result` status to `succeeded`. The code will give an overview of various methods in the custom policy step.

```
public class PacktpubCustomStep extends AbstractStep{
        public PacktpubCustomStep() {
    }
    public void init() throws IllegalStateException {
        // nothing to initialize
    }
    public void destroy() {
        // do nothing
    }
/* Custom policy step execute method implementation
*/
    public IResult execute(IMessageContext messageContext) throws
Fault {
        Result result = new Result();
        result.setStatus(IResult.SUCCEEDED); //initialize result
   return result;
    }
}
```

The result object type that the execute method returns has three main states, such as:

- IResult.FAILED
- IResult.SUCCEEDED
- IResult.SUSPENDED

Like the name indicates, each state describes the state of the custom policy step execution as either "failed" or "succeeded" or "suspended".

Now that the `AbstractStep` has been extended and the custom policy step code has been written, the next step is to create the step template XML file.

Deploying the Custom Policy Step

The custom policy step Java code should be compiled and converted as a jar file, and then it should be deployed. In order to compile the above Java program you should include the necessary JAR files, and then compile and create the new JAR file with a custom policy step. Once the custom policy step JAR file is created, it should be copied to `ORACLE_HOME/owsm/lib/custom`

Once the JAR files are copied, the Oracle WSM should be restarted for the new JAR file to take effect.

Step Template XML File Creation

Once the custom policy step extension code has been written, the next step is to deploy the code. As a part of the deployment process, create a step template XML file and upload them into Oracle Web Service Manager so that it can appear as one of the possible policy steps when you are ready to define the Policy to protect your web service.

The step template is nothing but an XML file that contains information such as:

- Name: Name of the step template.
- Id: A unique identifier for the step template.
- Package: Name of the Java package that will be executed when the policy step is invoked.
- Description: Description of the custom policy step.
- Implementation: Class name of the actual implementation that will be invoked when this step is executed.
- PropertyDefinitions: Contains a list of property definition sets.
- PropertyDefinitionSet: Contains a list of property definitions.
- PropertyDefinition: Contains information such as description, whether or not enabled, and any default value.

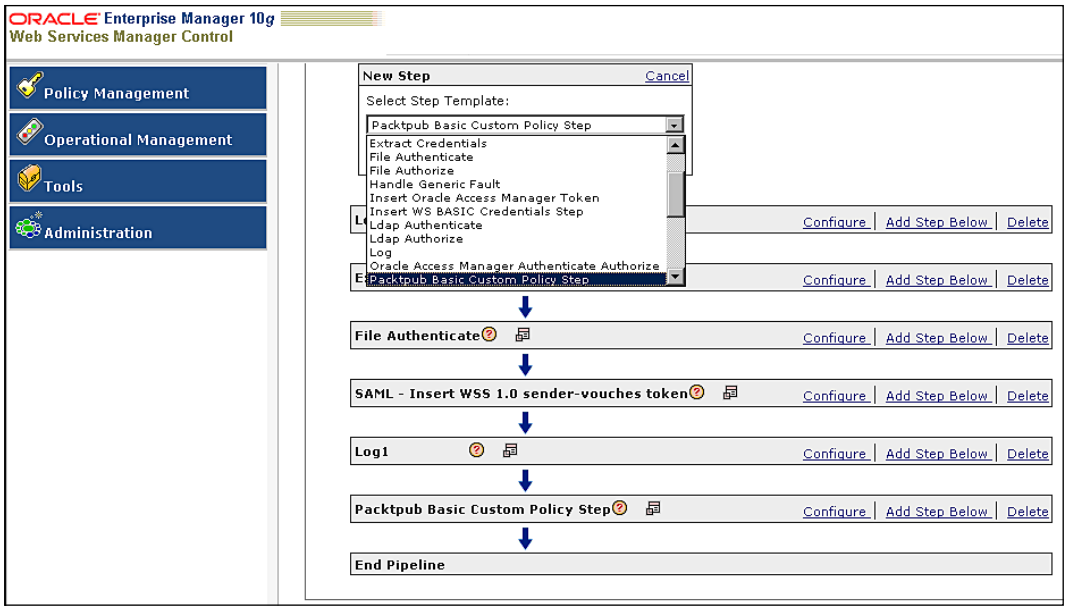
The sample XML file below describes the step template for the custom policy step that contains the above-mentioned information.

```
<csw:StepTemplate xmlns:csw="http://schemas.confluentsw.com/
ws/2004/07/policy" name="Packtpub Basic Custom Policy Step"
package="customsteps" timestamp="Oct 31, 2005 05:00:00 PM" version="1"
id="118970813">
   <csw:Description>PacktPub Sample Custom Policy Step -Basic Version
</csw:Description>
   <csw:Implementation>customsteps.PacktpubCustomStep</csw:
Implementation>
   <csw:PropertyDefinitions>
         <csw:PropertyDefinitionSet name="Basic Properties">
               <csw:PropertyDefinition name="Enabled"
type="boolean">
                     <csw:Description>If set to true, this step is
enabled</csw:Description>
                     <csw:DefaultValue>
                           <csw:Absolute>true</csw:Absolute>
                     </csw:DefaultValue>
               </csw:PropertyDefinition>
         </csw:PropertyDefinitionSet>
   </csw:PropertyDefinitions>
</csw:StepTemplate>
```

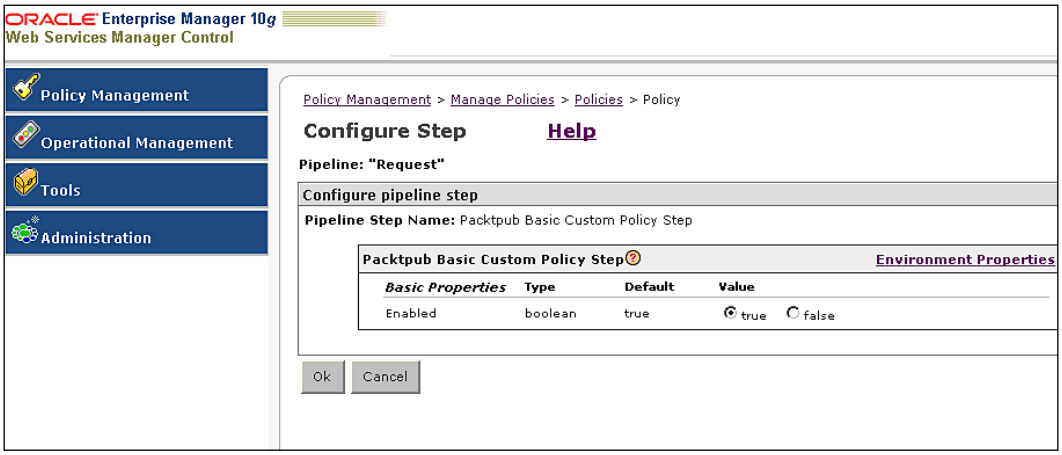
The above XML file is the step template for the packtpub custom policy step. The XML file contains information such as:

- `id`: Unique template Id as `118970813`
- `name`: `Packtpub Basic Custom Policy Step`
- `package`: `customsteps`
- `Implementation`: `customsteps.PacktpubCustomstep`

Once the step template is uploaded into Oracle WSM by clicking the **Add New Step** button (Navigate from **Manage Policies**, to click on **Steps** and then **Add New Step**), the policy step is available to be used in any policy. The following screenshot shows that the **Packtpub Basic Custom Policy Step** is available as one of the possible policy steps when you are defining a new policy for the web service.



Once you select the packtpub policy step, and click on the **Configure** tab, you will see the description of the packtpub policy step along with the property definitions that we included in the step template XML. The following screenshot shows the Packtpub custom step configuration window.



The description and the details that you see in the screenshot actually came from the step template XML that was uploaded.

Once the custom policy step is configured, the policy can be committed. You can then run a test to check if your customization actually took effect or not.

So far we have seen a very simple custom policy step that goes through the process of creating the custom implementation, deploying the JAR file, defining the step template and adding the new policy step as a part of the policy. While the above custom policy step just simply returns success without performing any action, you can write code to perform your custom implementations.

In the next section, we will walk through a detailed example of creating a custom policy step that will restrict access to the service only from a specified IP address, and will only give access to the configured web service method name.

Custom Policy Step Example: Restrict Access Based on IP Address to the Specified Method

In the last section, we walked through the steps to create and deploy a basic custom policy step. In this section, we will walk through an example of custom policy step which we will restrict access to the specific IP address and to the specific web service method.

Oracle Web Service Manager can only protect the web services and can enforce access control policies for the entire web service. However, Oracle WSM cannot restrict access only to specific web service methods within one web service. Also, there is no policy step available that will restrict access to web services based on the IP address. In this section, we will create a custom policy step that will:

- Grant access only to the specific web service method.
- Grant access only to the specific IP address.

In order for the custom policy step to restrict access based on an IP address, and only to a specific web service method, the custom policy step should know about:

- What web service method is allowed access for the specified policy?
- What IP address is a valid IP address that can be allowed to access the web service?
- What web service method is currently being invoked by the client application?
- What is the IP address of the client application?

The above questions can be categorized into two different sets. One set of questions are the configurable values within the policy step, i.e. what IP address is allowed access and what web service method can be executed, and what other set of values are available when the policy step is executed, i.e. which method is currently being executed and from where (IP address) the web service is being invoked.

The custom policy step has to simply take the configured parameter values from the policy step and compare that against the values that are available when the policy step is being executed. If they both match, the authorization is successful, otherwise access is denied.

In order to implement the above scenario, the custom policy step should consist of:

- Custom policy step Java code that extends the `AbstractStep` class and implements the `execute` method.
- Define the step template with parameters such as IP address and the web service method name.

Extending the AbstractStep

The `CustomStep` class extends the `AbstractStep` class and implements the `execute` method. This example is different from the previous example because here we will have to validate the IP address of the client application and the web service method name being invoked against the one that will be configured during the policy creation. The "shell" of the `customstep` class will look like:

```
public class CustomStep extends AbstractStep{
    private static String CLASSNAME = CustomStep.class.getName();
    private static ILogger LOGGER = LogManager.getLogger(CLASSNAME);
    private String allowedIpAddress = null;
    private String allowedRoleName = null;
    private String protectedServiceMethodName = null;

    public CustomStep() {
    }
    public void init() throws IllegalStateException {
        // nothing to initialize
    }
    public void destroy() {
        // do nothing
    }

    /**
```

```
     * This is the main method which will validate that the request is
coming from
     * the correct IP Address and has permission to access the
specified metod.
     */
    public IResult execute(IMessageContext messageContext) throws
Fault {
        LOGGER.entering(CLASSNAME, "execute");

        Result result = new Result();
        result.setStatus(IResult.FAILED); //initialize result
   return result;
}
}
```

In the above example, the `execute` method was just returning the `Result` object with the status as **Failed**. However, we need to add functionality to the code to:

- Execute the policy step only during the request pipeline.
- Retrieve the IP address of the remote host.
- Retrieve the web service method name being executed.

In order to add the additional functionality, it is important to know where to get that information from. The `execute` method of the custom policy step has a parameter of type `IMessageContext`.

While `IMessageContext` has various methods and fields, for our example, we are interested in knowing how to retrieve:

- Current processing stage
- Remote host IP address
- Web service method name being executed

The following code snippets describe how to get the remote host IP address, web service method name, etc.

```
        String processingStage = messageContext.getProcessingStage();
```

The `getProcessingStage` method will return the processing state information. You can then compare this with the static fields of the `IMessageContext` to find out if the processing is at request or response stage.

```
        boolean isRequest =
                (IMessageContext.STAGE_REQUEST.equals(messageContext.
getProcessingStage()) ||
                IMessageContext.STAGE_PREREQUEST.
equals(messageContext.getProcessingStage()));
```

The previous code snippet compares the value returned from `getProcessingStage` with the static fields `STAGE_REQUEST` and `STAGE_PREREQUEST`.

The following code snippet will return the web service method name being executed and also the remote host IP address.

```
         MessageContext msgCtxt = (MessageContext) messageContext;
           String _MethodName = msgCtxt.getRequest().getMethodName();
        LOGGER.log(Level.INFO, "Writing Allowed IP Addr before creating
SOAP header " + msgCtxt.getRemoteAddr() );
```

Now that we have added functionality to retrieve the remote host IP address and the web service method name, we now have to define and get the IP address is that allowed to access, and the web service method that is granted access to the client.

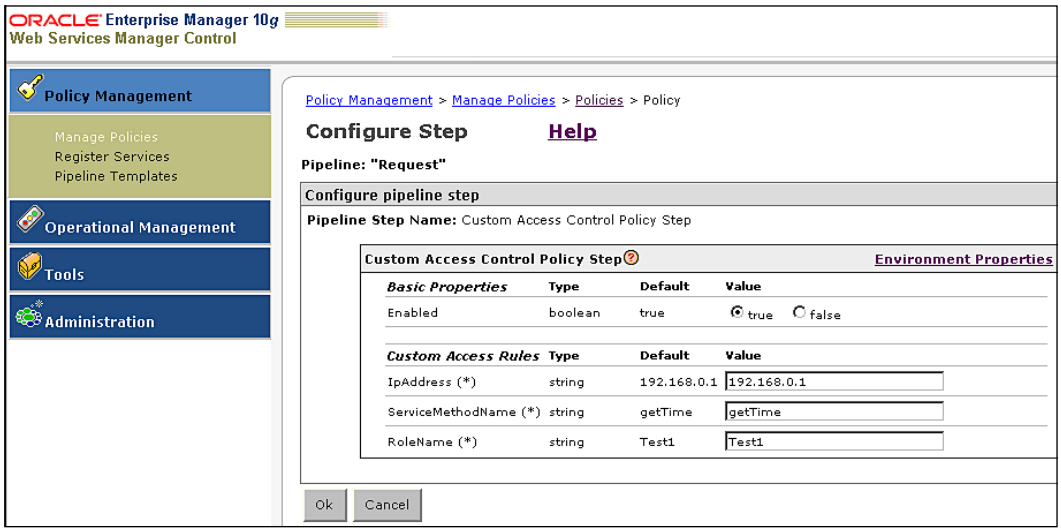
We have to first define the IP address and the web service method name as configurable parameters in the policy step. This can be done by creating the step template XML file with both the IP address and web service method name as parameters. The step template XML file will also include the unique template ID, name, implementation class, etc. The following is the step template XML file.

```
<csw:StepTemplate xmlns:csw="http://schemas.confluentsw.
com/ws/2004/07/policy" name="Custom authenticate step"
package="customsteps" timestamp="Oct 31, 2005 05:00:00 PM" version="1"
id="118970810">
   <csw:Description>Custom step that authenticates the user against
the credentials entered here. This step requires Extract credentials
to be present before it in the request pipeline.</csw:Description>
   <csw:Implementation>customsteps.CustomStep</csw:Implementation>
   <csw:PropertyDefinitions>
         <csw:PropertyDefinitionSet name="Basic Properties">
               <csw:PropertyDefinition name="Enabled"
type="boolean">
                     <csw:Description>If set to true, this step is
enabled</csw:Description>
                     <csw:DefaultValue>
                           <csw:Absolute>true</csw:Absolute>
                     </csw:DefaultValue>
               </csw:PropertyDefinition>
         </csw:PropertyDefinitionSet>
         <csw:PropertyDefinitionSet name="Custom Access Rules">
               <csw:PropertyDefinition name="IpAddress"
type="string" isRequired="true">
                     <csw:DisplayName>IpAddress</csw:DisplayName>
                     <csw:Description>IP Address that is allowed
access</csw:Description>
                     <csw:DefaultValue>
                           <csw:Absolute>192.168.0.1</csw:Absolute>
                     </csw:DefaultValue>
```

```
                    </csw:PropertyDefinition>
                    <csw:PropertyDefinition name="ServiceMethodName"
type="string" isRequired="true">
                        <csw:DisplayName>ServiceMethodName</csw:
DisplayName>
                        <csw:Description>Service Method Name that is
Protected (Secured)</csw:Description>
                        <csw:DefaultValue>
                            <csw:Absolute>getTime</csw:Absolute>
                        </csw:DefaultValue>
                    </csw:PropertyDefinition>
                </csw:PropertyDefinitionSet>
            </csw:PropertyDefinitions>
</csw:StepTemplate>
```

In the example above, there are two property definitions: one for the `IpAddress` and another for `ServiceMethodName`. Once this template is uploaded and added as a new step, these properties will be available to be configured when defining the policy.

The following screenshot shows the list of available parameters that can be configured once the custom policy step is added to the web service policy.



Now the next step is to make sure that the custom policy step code can read the values that are configured during the policy creation. In order for the custom code to read the values, the code should have the method (properties) defined with the same name as the one used in the step template XML file.

In our example, the step template XML had two property definitions, one called as `IpAddress` and the other called as `ServiceMethodName`.

```
            <csw:PropertyDefinitionSet name="Custom Access Rules">
                  <csw:PropertyDefinition name="IpAddress"
type="string" isRequired="true">
                        <csw:DisplayName>IpAddress</csw:DisplayName>
                        <csw:Description>IP Address that is allowed
access</csw:Description>
                        <csw:DefaultValue>
                              <csw:Absolute>192.168.0.1</csw:Absolute>
                        </csw:DefaultValue>
                  </csw:PropertyDefinition>
                  <csw:PropertyDefinition name="ServiceMethodName"
type="string" isRequired="true">
                        <csw:DisplayName>ServiceMethodName</csw:
DisplayName>
                        <csw:Description>Service Method Name that is
Protected (Secured)</csw:Description>
                        <csw:DefaultValue>
                              <csw:Absolute>getTime</csw:Absolute>
                        </csw:DefaultValue>
                  </csw:PropertyDefinition>
            </csw:PropertyDefinitionSet>
```

The custom policy step code should have the get and set methods that have the same name as the one defined in the `name` attribute of the `PropertyDefiniton` element. In our example, the `IpAddress` and `ServiceMethodName` are the names that are defined in the step template. The custom policy code should have `getIpAddress`, `setIpAddress`, `getServiceMethodName`, `setServiceMethodName` to retrieve the values that will be configured during the policy creation. The following examples describe the get and set methods:

```
public String getIpAddress() {
    return allowedIpAddress;
}
public void setIpAddress(String IpAddress) {
    this.allowedIpAddress = IpAddress;
}
public String getServiceMethodName() {
    return protectedServiceMethodName;
}
public void setServiceMethodName(String serviceMethodName) {
    this.protectedServiceMethodName = serviceMethodName;
}
```

Now that we have retrieved the remote host IP address and the web service method name being executed, and also the configured values from the policy, we now have to compare those values to find out if they are the same. The following code describes how the two `IpAddress` values and the web service method names are compared and the result object is set accordingly.

```
          if (allowedIpAddress.equals(msgCtxt.getRemoteAddr()) && _
MethodName.equals(protectedServiceMethodName) )
                {
                      result.setStatus(IResult.SUCCEEDED);
                }
          else
                {

   msgCtxt.getInvocationStatus().setAuthorizationStatus(InvocationSta
tus.FAILED);

                }
```

Now that we have looked at the various components of writing and deploying custom policy step, let's take a minute to summarize the steps to develop the custom policy step. In a nutshell one has to:

- Design the custom policy and decide how many parameters will be accepted and their data types.
- Write a Java code that extends the `AbstractStep`.
- Implement the `execute` method.
- Override `Init` and `Destroy` if required.
- Create the step template XML file that contains the property names for the parameters.
- Add properties (methods) that match property name with the appropriate data type.
- Complete the execute method.
- Compile the Java code.
- Create the JAR file.
- Copy the JAR file to `/owsm/lib/custom`.
- Upload the step template XML file.
- Create or modify policy and include the policy step appropriately.
- Test the web service and ensure that the custom policy is invoked.

The example below shows the complete custom policy step code that does the following:

- Execute only during the request processing stage.
- Get the remote host IP address and the executing web service method name.
- Get the parameter values entered during the policy creation.
- Compare the remote host IP address with the configured value and compare the executing web service method name with the configured value.
- Return the result object with appropriate status.

```
/*
 * CustomStep.java
 *
 * This custom policy step performs authorizatio check to ensure
 * that the service is invoked from a specific IP Address and
 * only the specified web service method.
 *
 */
package customsteps;

import com.cfluent.ccore.util.logging.*;
import com.cfluent.pipelineengine.container.MessageContext;
import com.cfluent.policysteps.sdk.*;
import java.io.ByteArrayOutputStream;
import java.util.Locale;
import javax.xml.soap.SOAPFactory;
import org.apache.axis.message.SOAPEnvelope;
import org.apache.axis.message.SOAPHeaderElement;
public class CustomStep extends AbstractStep{
    private static String CLASSNAME = CustomStep.class.getName();
    private static ILogger LOGGER = LogManager.getLogger(CLASSNAME);
    private String allowedIpAddress = null;
    private String allowedRoleName = null;
    private String protectedServiceMethodName = null;

    public CustomStep() {
    }
    public void init() throws IllegalStateException {
        // nothing to initialize
    }
    public void destroy() {
        // do nothing
```

```
    }
    /**
     * This is the main method which will validate that the request is
coming from
     * the correct IP Address and has permission to access the
specified metod.
     */
    public IResult execute(IMessageContext messageContext) throws
Fault {
        LOGGER.entering(CLASSNAME, "execute");
        Result result = new Result();
        result.setStatus(IResult.FAILED); //initialize result
        String processingStage = messageContext.getProcessingStage();
        LOGGER.log(Level.INFO, "Processing stage is " +
processingStage);
        boolean isRequest =
                (IMessageContext.STAGE_REQUEST.equals(messageContext.
getProcessingStage()) ||
                IMessageContext.STAGE_PREREQUEST.
equals(messageContext.getProcessingStage()));
   //Execute the step Only when its a Request pipeline else return
success
        if(!isRequest) {
            result.setStatus(IResult.SUCCEEDED);
            return result;
        }
        MessageContext msgCtxt = (MessageContext) messageContext;
          String _MethodName = msgCtxt.getRequest().getMethodName();
        LOGGER.log(Level.INFO, "Writing Allowed IP Addr before
creating SOAP header " + allowedIpAddress );
       LOGGER.log(Level.INFO, "Writing Allowed IP Addr before creating
SOAP header " + msgCtxt.getRemoteAddr() );
          if (allowedIpAddress.equals(msgCtxt.getRemoteAddr()) && _
MethodName.equals(protectedServiceMethodName) )
                {
                       result.setStatus(IResult.SUCCEEDED);
                }
          else
                {
                      msgCtxt.getInvocationStatus().setAuthorization
Status(InvocationStatus.FAILED);
                }
```

```
// Set the result to SUCCESS
//result.setStatus(IResult.SUCCEEDED);
return result;
    }

    public String getIpAddress() {
        return allowedIpAddress;
    }
    public void setIpAddress(String IpAddress) {
        this.allowedIpAddress = IpAddress;
        LOGGER.log(Level.INFO, "IP Address is.. " + allowedIpAddress);
    }
    public String getServiceMethodName() {
        return protectedServiceMethodName;
    }
    public void setServiceMethodName(String serviceMethodName) {
        this.protectedServiceMethodName = serviceMethodName;
    }
    public String getRoleName() {
        return allowedRoleName;
    }
    public void setRoleName(String roleName) {
        this.allowedRoleName = roleName;
    }
}
```

Now that the custom policy step code is created, the next step is to compile the code. In order to compile the above code, you need to include the following JAR files:

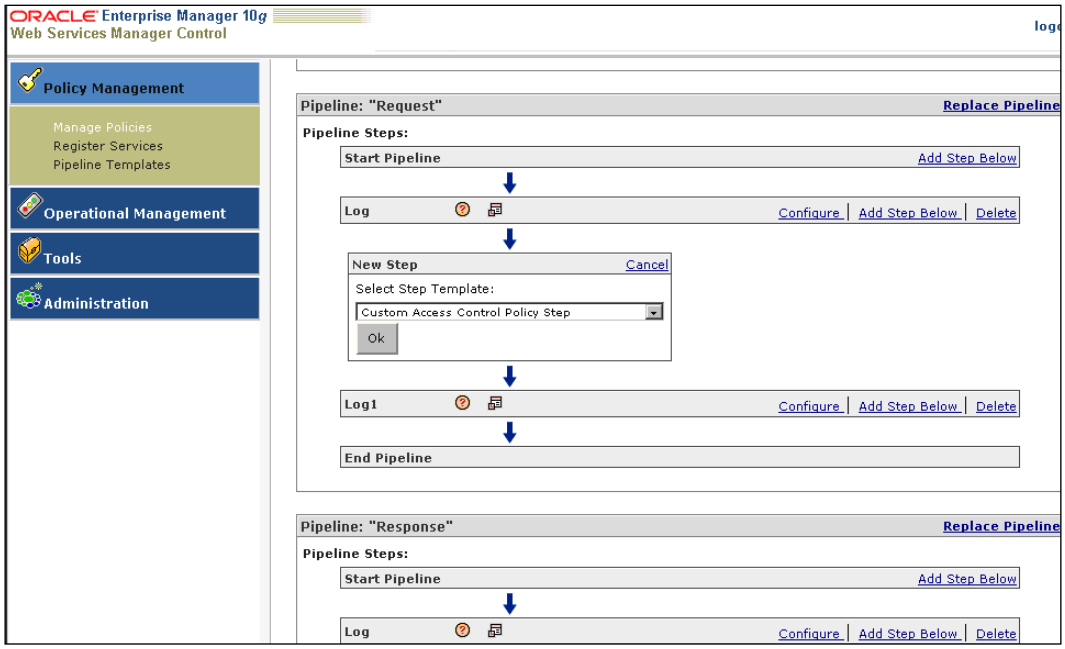
```
/owsm/lib/extlib/saaj.jar
/owsm/lib/extlib/axis.jar
/owsm/lib/coresv-4.0.jar
```

Once the `CustomStep.java` is compiled, the `CustomAuthenticationStep.JAR` file can be created and copied to `/owsm/lib/custom/CustomAuthenticationStep.jar`. The **ant** build file is attached along with the source code for this chapter.

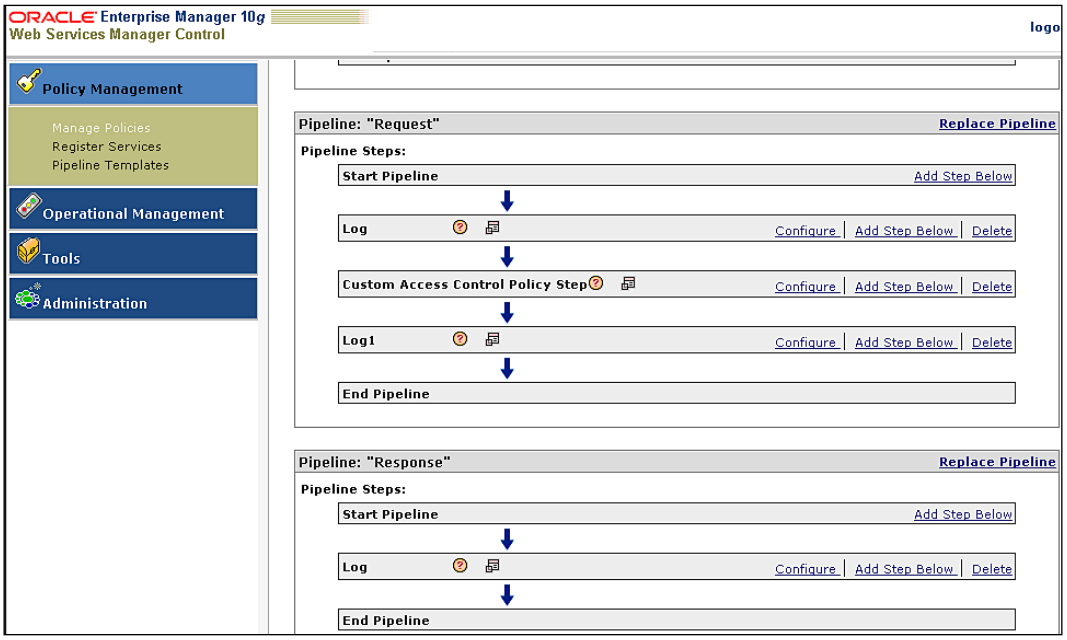
Testing the Custom Policy Step

Once the custom policy step is deployed and the step template is uploaded, the custom policy step is available to be included in any policy step. Let's take an existing web service policy and add the new custom policy step so that only specific IP addresses are allowed access to the web service method that is configured.

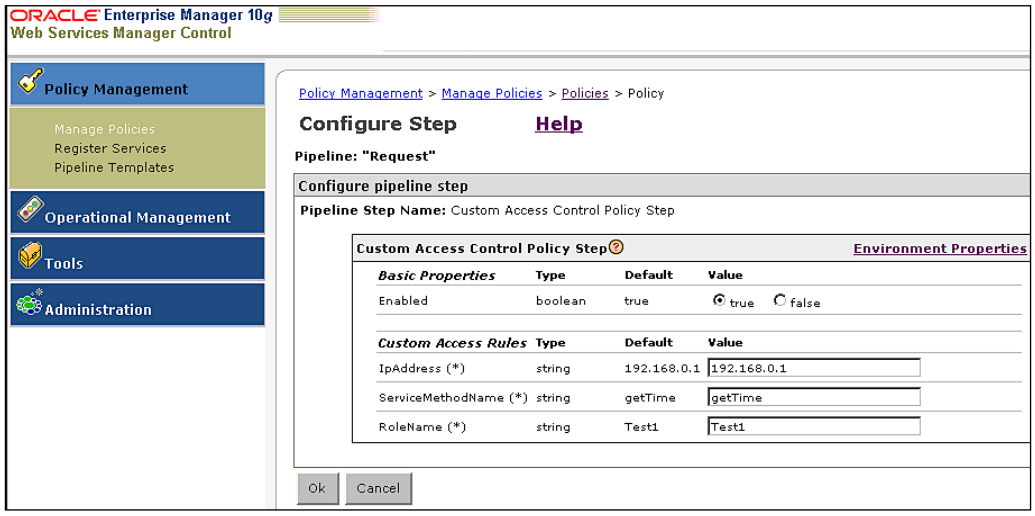
In order to enforce the custom authorization rules, edit the policy for the web service and in the **Request** Pipeline, click **Add Step Below** to select the custom policy step which is to be added to the policy (refer to the following screenshot).



Once you click **Ok**, you will see that the custom policy step is added to the policy as shown in the following screen capture.



Now you can click on the **Configure** button to enter the **IpAddress** value and the web service method name. When you click **Configure**, you will see the screen to enter the IP address and web service method name as shown in the following screen capture.



Once you click **Ok** and then save the policy, the new custom step is added to the policy.

In order to test the custom policy step, go to the **Tools** section and then select **Test Page** and enter the WSDL for the service. When you invoke this service, it will grant access only if the configured IP address and the remote host address matches, and also the configured web service method name and the method being invoked matches. If the custom policy step executes successfully you will get the web service response message.

In the above example, we discussed in detail about how to create the custom policy code, how to deploy the jar file and also how to create the step template XML file. The source code is attached with this chapter and the jar file can be deployed to get a quick start on the custom policy step.

Summary

In this chapter, we took a closer look at how to add a custom policy step to enforce any custom authentication or authorization rules. Custom policy steps can be built to address any type of customization that is not directly addressed by Oracle WSM, such as validating **UserNameToken**, validating IP address of the consumer, etc. Oracle Web Service Manager can be used to externalize the web service security policy definition and enforcement so that the web service and the consumer applications are relieved from performing the underlying security functionalities.

8
Deployment Architecture

Oracle Web Services Manager enforces the security policies for web services and it is critical that the Oracle WSM components are highly available and scalable to meet any service level agreements. In order to design a highly available and scalable architecture, one has to understand the various components of Oracle WSM and how they can be deployed individually or in combination to meet the requirements. In this chapter, I will discuss the various components of Oracle WSM and how they can be deployed to ensure high availability and scalability.

Oracle WSM Components

The main functionality of the Oracle Web Services Manager is to give the administrator the ability to define the security policy, define service level agreements, and configure Oracle WSM to enforce security policy and to monitor the SLA. In order to perform the above functions, the Oracle WSM leverages few of its internal components, such as:

- Policy Manager—Used to register web services, define the security policy, etc.
- Policy Enforcement Point—EP will enforce the security policy deployed either as a gateway or as an agent.
- Oracle WSM Monitor—To monitor the events and SLAs, etc.
- Oracle WSM Database—Where all the policy data and configurations are stored.
- Web Services Manager UI Control—Web UI to administer the Oracle WSM.

In the deployment of Oracle WSM, both the scalability and high availability should be considered.

Addressing Oracle WSM Scalability

Oracle Web Services Manager can be deployed to handle a growing web services environment, and the components of Oracle WSM can be deployed on one or many computers to handle the demand. In addressing the scalability of Oracle WSM, the first and foremost is the policy enforcement point. The policy enforcement point can be either a Oracle WSM gateway or an agent. The agents, i.e. server agents or client agents, can be attached to the application platform (client or server) to maximize the performance of PEP. The Oracle WSM gateways can also be deployed in different servers and each talking to the same policy manager to aggregate a certain set of services with one set of gateways to maximize the performance and also to address the high availability and SLAs. The PEP can be deployed across many computers to distribute the load to increase the performance.

While PEP can be deployed to optimize the performance of enforcing the security, the policy manager can also be deployed across multiple computers, and behind a load balancer to ensure high availability and to optimize performance to register and define web services policies.

Note: When load balancer is used in front of the policy manager, the load balancer should be configured to have the "sticky bit" on so that it maintains the session with a particular Oracle WSM component on the same server for the same client request.

Addressing High Availability

Oracle Web Services Manager should be deployed in such a way that it is highly available to meet the customer requirements, which in some cases may be 24 by 7. The simple answer to high availability is to configure the Oracle WSM components behind a load balancer. In order to configure Oracle WSM behind a load balancer, the Oracle WSM components should be installed and configured in a specific order on multiple machines, as described in this section.

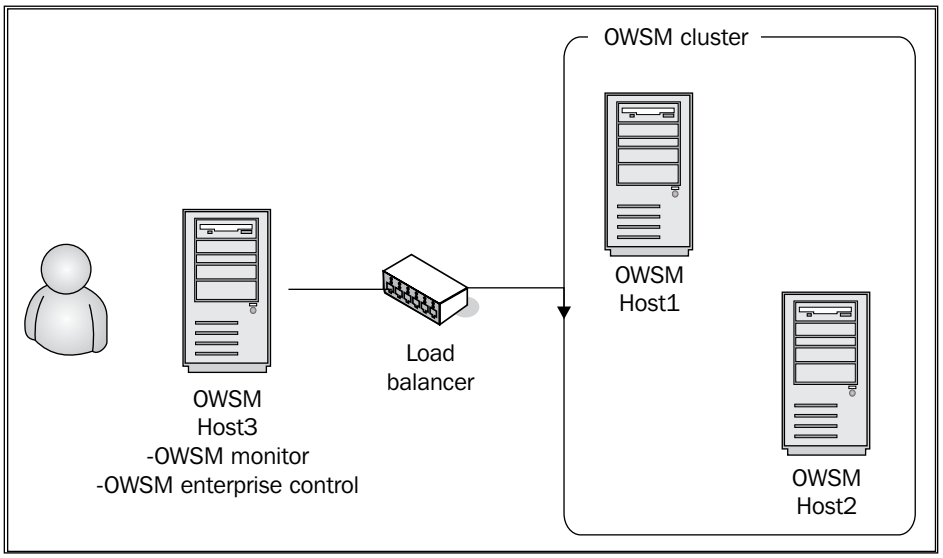
In a typical load balanced configuration, you would deploy Oracle WSM components on two different hosts and then configure the load balancer to send the requests to those two hosts. However, with Oracle WSM, you have to handle the component ID, which is unique to each Oracle WSM installation. In order to have Oracle WSM load balanced, the component ID should be unique when the request goes to any of the Oracle WSM installations. The following steps describe how to configure Oracle WSM in a load balanced environment.

Installation

The first step is to install the Oracle WSM components on three different hosts:

- Install Oracle WSM components on Host1
- Install Oracle WSM components on Host2
- Install Oracle WSM components on Host3
- Configure load balancer to send requests to Host1 and Host2

We will use Host1 and Host2 behind the load balancer which contains the Oracle WSM policy manager of the Oracle WSM gateway. The Host3 will contain the Oracle WSM monitor and the Oracle WSM web services manager control. The following diagram shows the overview of the Oracle WSM highly available architecture.



Once the Oracle WSM components are installed on all the three hosts, the Host3 which contains the web services manager control should be configured to talk to the load balancer (so that the requests go to Host1 or Host2).

On the Host3, you should edit the `ui-config-installer.properties` file to specify the host name of the load balancer. The following configuration changes should be made on the `ui-config-installer.properties` file.

Open the `ui-config-installer.properties` file under `ORACLE_HOME/owsm/config/ccore/`.

Add or modify the following properties to change the host name and port number:

- `Ui.pm.server.httphost`—Host name of the load balancer which manages the Oracle WSM policy manager.
- `Ui.pm.server.httpport`—Port number of the load balancer which maps to Oracle WSM policy manager.

Once you modify the properties file, you should run the following commands to deploy the solution.

```
Wsmadmin deploy control
```

Disabling Unnecessary Components

It is clear from the previous figure that each host only needs a few components, but each Oracle WSM default installation comes up with all the necessary components. From the previous figure, we need the following configuration:

- Host1—Oracle WSM policy manager and gateway
- Host2—Oracle WSM policy manager and gateway
- Host3—Oracle WSM monitor and web services manager control

So in all the three hosts, we should remove the components that are not required, and you can do that by using the following commands in each environment.

Host1 and Host2:

On Host1 and Host2 we need only the Oracle WSM policy manager and gateway. We need to uninstall the monitor and the web services manager control and you can do that by running the following commands on each host.

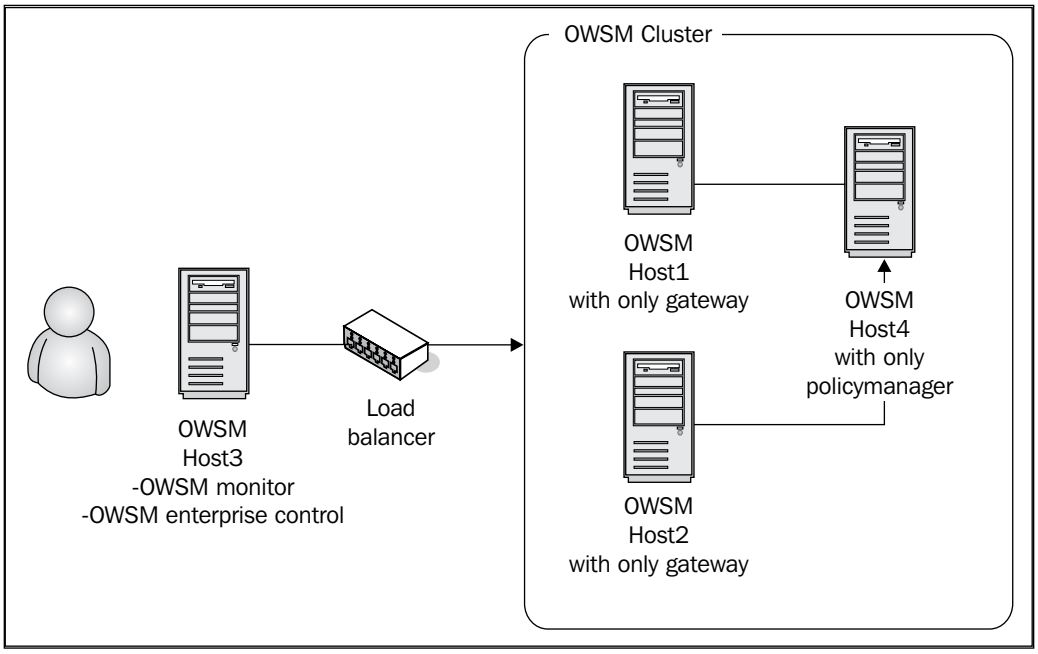
```
wsmadmin undeploy monitor
wsmadmin undeploy control
```

Host3:

On Host3 you need to uninstall the policy manager and the gateway. You can do that by running the following commands.

```
wsmadmin undeploy policymanager
wsmadmin undeploy gateway
```

Note: It is not necessary to deploy both the policy manager and gateway on the same host. You can have multiple gateways talking to the same policy manager. Typically the load is on the gateway and not on the policy manager. Deploying policy manager and gateway on two different machines will also make it more secure. In order to remove the policy manager from a host, you can undeploy policy manager using the `wsmadmin` command. The following figure shows how you can have two gateways and one policy manager and one monitor/UI control.



Mapping Component ID on Host1 and Host2

When you invoke Oracle WSM from Host3, which points to one of the Oracle WSM on Host1 or Host2 via the load balancer, you need to make sure that both hosts have the same component ID for the gateway. You can register a gateway by connecting to the Oracle WSM policy manager from Host3 through the web services manager control. When you register a gateway, it will generate a component id and will also ask for the URL of the component. When asked for the URL of the component, enter the load balancer URL. Once registered, the component ID is generated.

Now we need to make sure that the same component ID is in the `gateway-config-installer.properties` file on both Host1 and Host2. The `gateway-config-installer.properties` can be found at `ORACLE_HOME/owsm/config/gateway/`.

Make sure that the property `gateway.component.id` is the one that you just generated, otherwise replace the value with the one you just generated.

```
Example: gateway.component.id =C0003001
```

Once the changes are made, you need to deploy the gateway by executing the following command:

```
wsmadmin deploy gateway
```

Configuring Oracle WSM Monitor on Host3

Now that Host1 and Host2 have Oracle WSM gateway and policy manager, and Host3 has the Oracle WSM monitor, we need to configure Oracle WSM monitor on Host3 to monitor for events on both Host1 and Host2. You can edit the gateway properties on each host and modify the `cfluent.monitor.rmi.host` and `cfluent.monitor.rmi.port` values to map to Host3 where the Oracle WSM monitor is deployed.

Summary

Oracle Web Services Manager is designed to be scalable to address the growing web services environment and can be highly available by configuring behind a load balancer. In this chapter, we took a closer look at how to create a clustered Oracle WSM deployment and how easy it is to horizontally scale the Oracle WSM components.

9
Oracle WSM Runtime-Monitoring

Oracle Web Service Manager enforces security policies for the web service and hence it is critical to know about ongoing security violation, authentication failure, authorization failures, any delay in service, etc. Oracle WSM is a critical component of SOA infrastructure and requires appropriate auditing and monitoring to proactively address any service failures or security violations. In this chapter, we will discuss in detail how to manage the Oracle WSM environment from an operation point of view, i.e. how the monitoring works.

Oracle WSM Operational Management

Oracle Web Service Manager not only enforces the security policies for the web services, but also virtualizes the web service end points (vaguely like reverse proxy where you can access intranet web applications from outside of a corporate network). So when Oracle WSM is down or not operational, it's not just the security that is affected but also the service operations (that may or may not have any security attached to it). On the other hand, if you have a lot of web services, you can register them in Oracle WSM, which not only virtualizes the web service end points, but also monitors the health of the web services. It is important to monitor the various operations and statistics of the Oracle WSM to proactively address any operational issues. Oracle WSM has built-in functionality to monitor:

- Overall Statistics
- Security Statistics
- Service Statistics

Oracle Web Services Manager can also let you create custom views for the administrator to select and view certain statistics for all or few services. The administrator can also create **alarms** to notify them in case of any failure that the administrator is they'd be interested in.

Oracle WSM collects the statistics data at the gateway and at the agents which are then transferred to the Oracle WSM monitor, where the data is persisted to a database and then displayed as a report or pie chart view. Since the data regarding all the statistics can be huge, the it is automatically purged every 100 minutes. If you choose to keep the data for a longer period of time, you should modify the `monitor.aggregator.measurementStore.WindowSize` and then redeploy the Oracle WSM.

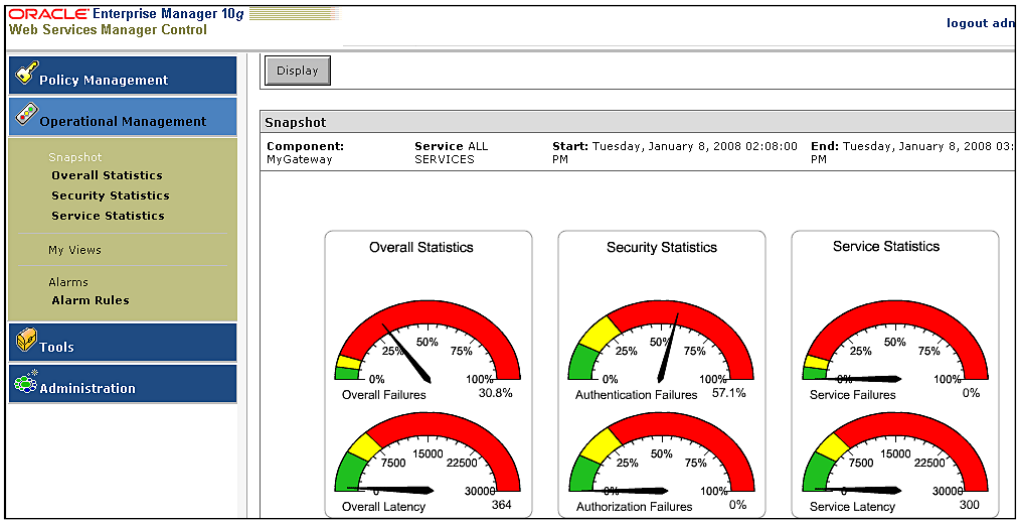
In the following sections, we will discuss each statistics in detail and also discuss how to create custom views and alarms.

Oracle WSM Overall Statistics

To begin with, administrators need to know the overall behaviour or statistics of Oracle Web Services Manager that will give more information about:

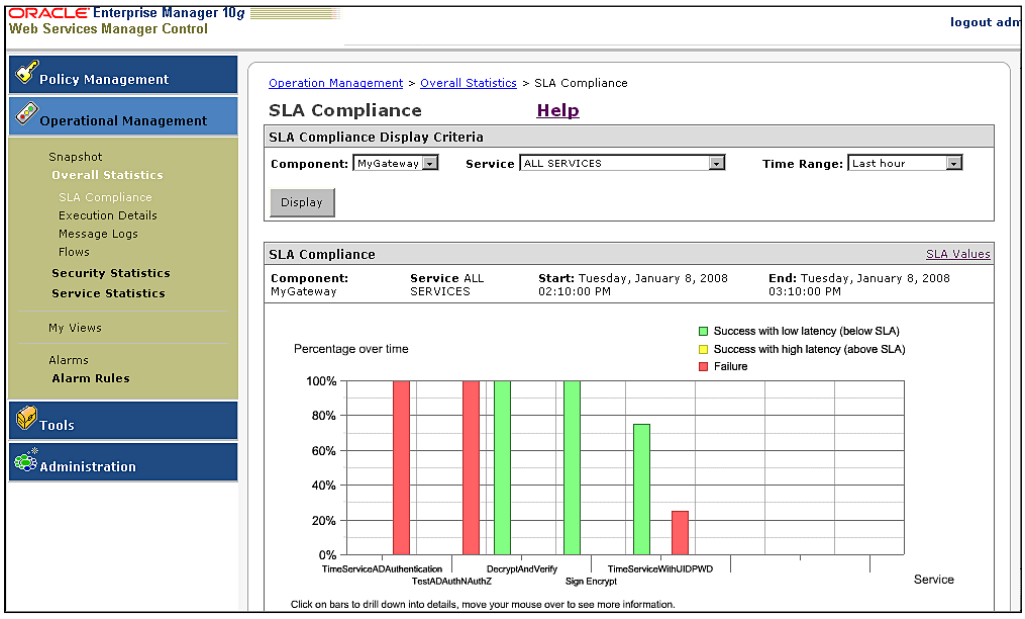
- Overall failures
- Authentication failures
- Service failures
- Overall latency
- Authorization failures
- Service latency

In Oracle WSM, you can go to **Operational Management** and click on **Snapshot** to get the overall statistics of all the web services in one place. Before you click on **Operational Management**, go to the **Test Page** under **Tools** and invoke a few web services with valid credentials and invalid credentials so that Oracle WSM can collect all the relevant data to display the report as shown in the following screenshot.



The figure above gives complete statistics such as 30% **Overall Failures** and 50% **Authentication Failures**.

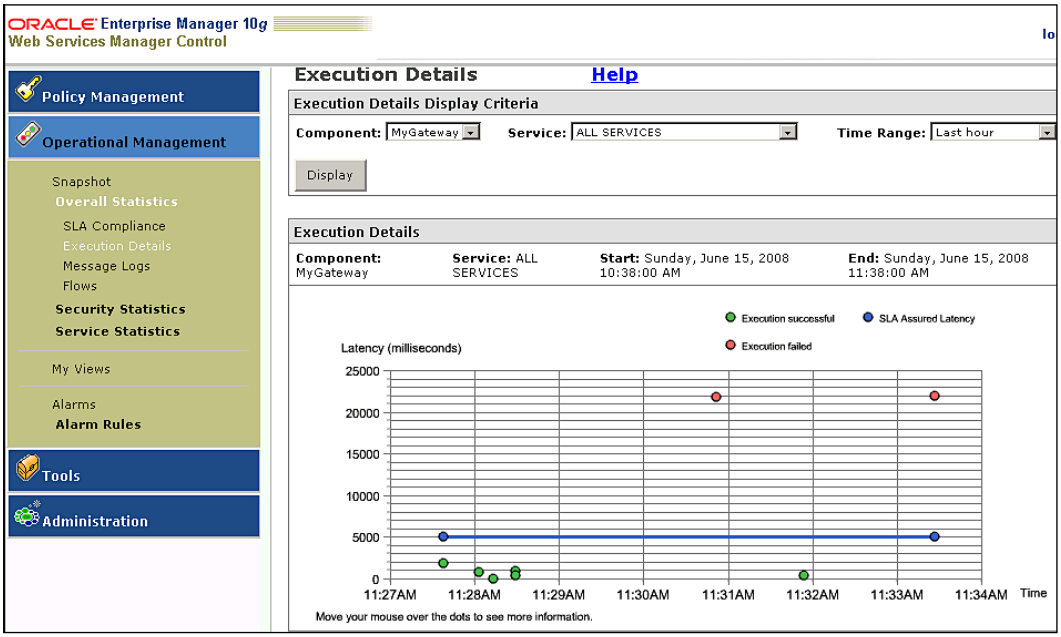
Once you identify that there are some failures, you may want to know which service operations did not meet the service level agreements. Oracle WSM gives the flexibility to define the SLA values and the administrators can actually monitor individual services that failed, or did not meet the latency requirements as shown in the following screenshot.



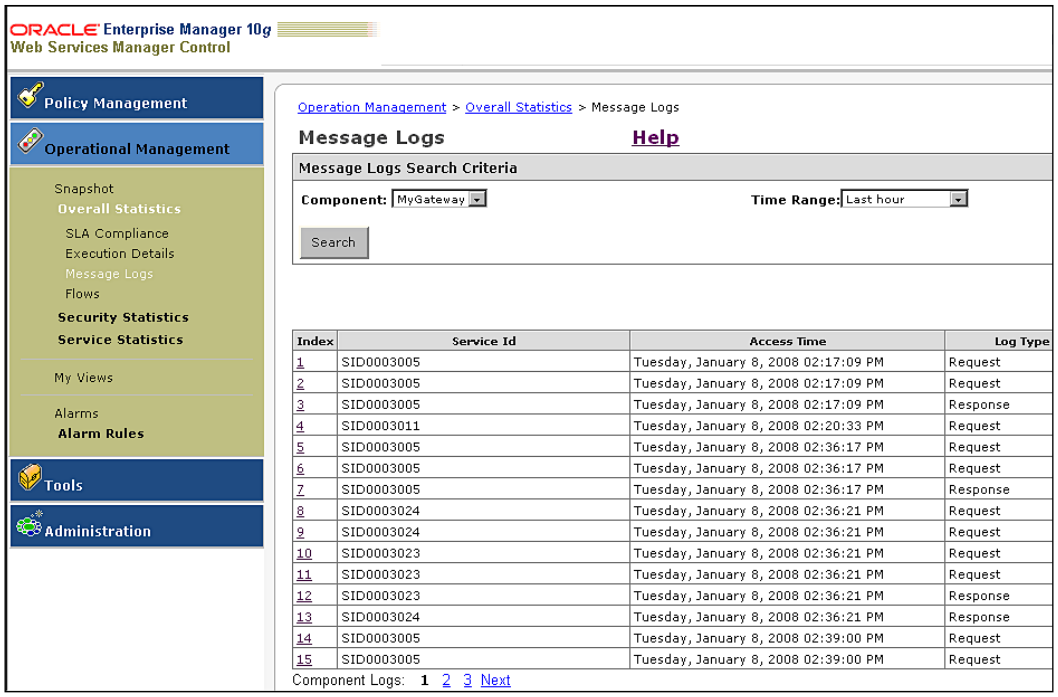
You can navigate to the **SLA Compliance** section from **Operational Management** by clicking on **Snapshot**, **Overall Statistics**, and then **SLA Compliance**. You can also modify the SLA values by clicking on the **SLA Values** hyperlink. You can also move the mouse over it to get more detailed information.

Now that you were able to monitor whether or not the web services meet the service level agreements defined, you can actually see the details of the web service execution by clicking on the **Execution Details** tab on the left-hand side menu. In the following screenshot, you can see the execution details for all the web services, where it indicates:

- **Latency** in milliseconds
- **Execution successful**
- Execution failure
- **SLA Assured Latency**



Once you are able to see the **Execution Details**, and if you are curious about execution success or execution failure and would like to see the message logs, you can click on the **Message Logs** from the left-side menu. This will take you to the list of message logs that was logged as shown in the following screenshot.



In the screenshot, you can see the list of message logs and you can click on the hyperlinks for individual services to look at request or response operations. In order to view the logs here, you should have added log steps in the Request or Response pipelines.

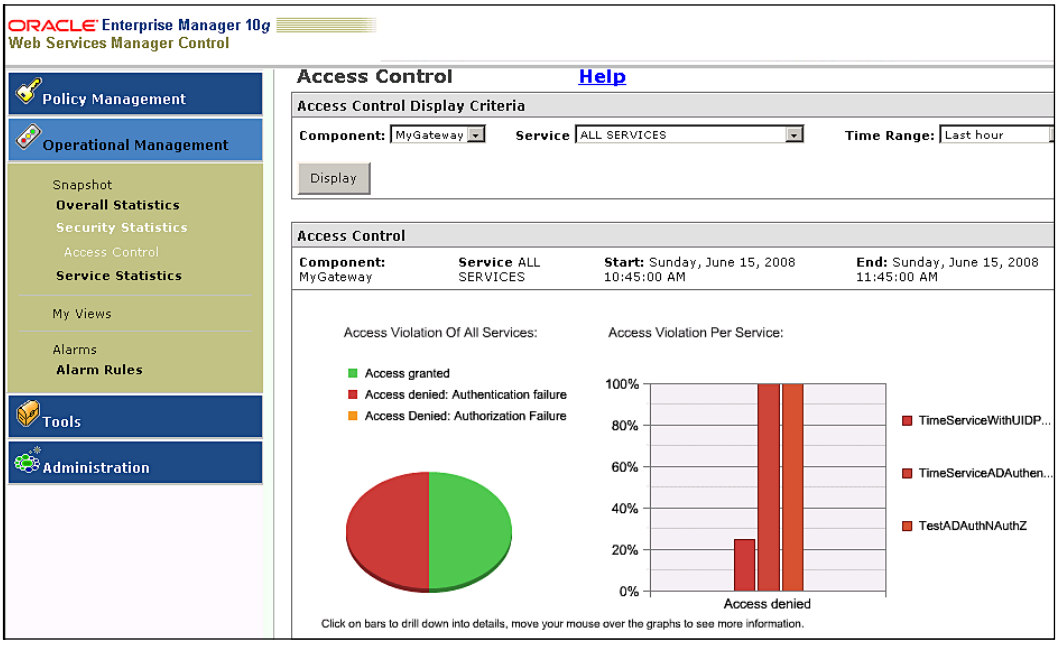
The overall statistics helps in maintaining the services as per the SLA. However, it is also critical to monitor the security statistics such as authentication failures and authorization failures to monitor any security failure or security breaches. In the next section, we will discuss more about the security statistics.

Oracle WSM Security Statistics

The operation management feature of Oracle WSM can also monitor the security statistics for the authentication and authorization failures. It is important to know how many web services failed at the authentication or authorization step. This will help address a couple of things:

- Service failure due to wrong credential or access privilege.
- Security breach, where an intruder is trying to access the system.

The administrator can navigate to **Security Statistics**, **Access Control** from the **Operational Management** to get a complete view of access control violations for all the services. When you click on **Access Control** from the left-side menu, you can see a bar diagram as shown in the following screenshot, which describes the percentage of **Access Denied**.

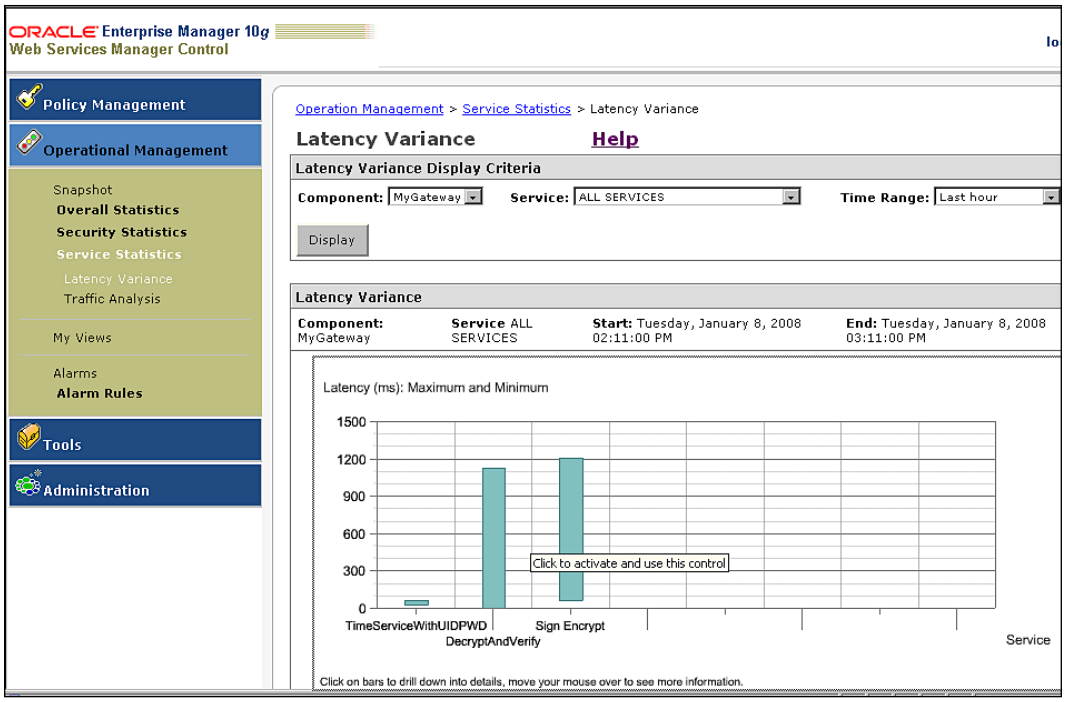


From the screenshot, you can get a clear picture of what percentage of each service were denied access, either because of improper authentication credentials or authorization privileges. You can also move the mouse over the pie chart to get the details.

Oracle WSM Service Statistics

While it is important to understand the overall statistics and the security statistics, it is also important to know how often a particular service is being invoked and what the overall latency for each service is. Oracle Web Services Manager can actually keep track of each service invocation and the latency.

One can navigate to **Service Statistics** from **Operational Management** and then click on **Latency Variance** to find out the minimum and maximum latency for each service during a particular period of time as shown in the following screenshot.

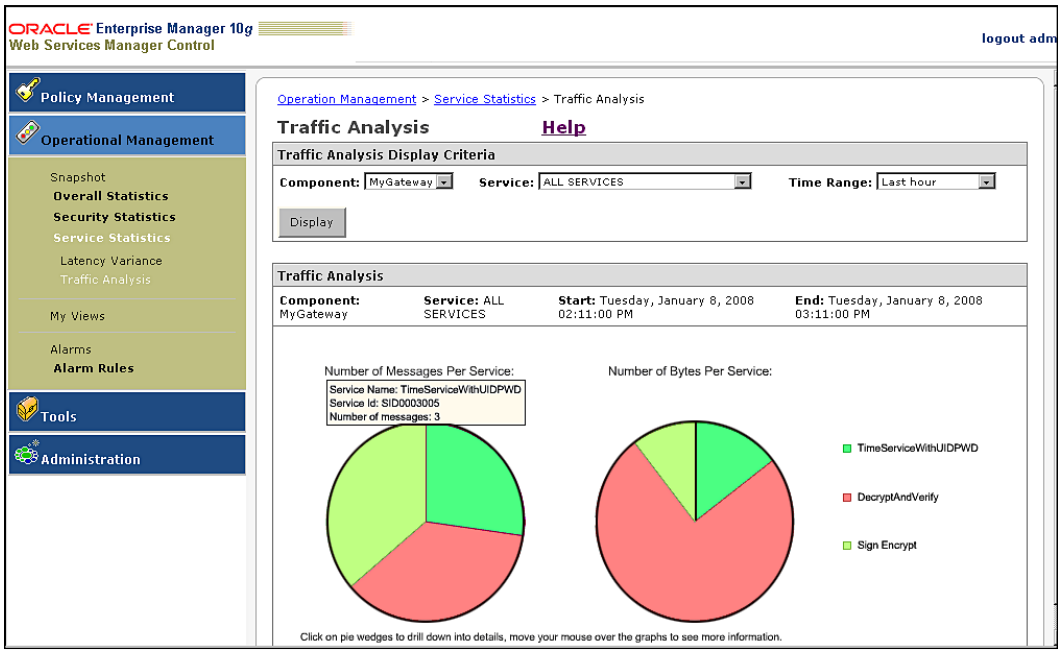


The figure shows the minimum and maximum latency for the services invoked in the last hour.

Both developers and administrators are also interested in finding out how often the web services were invoked in a particular period of time. You can navigate to the **Traffic Analysis** link to find out:

- How often a service was invoked in a particular time period.
- **Number of Bytes per Service** in a particular time period.

The following screen capture shows the traffic details of the services.



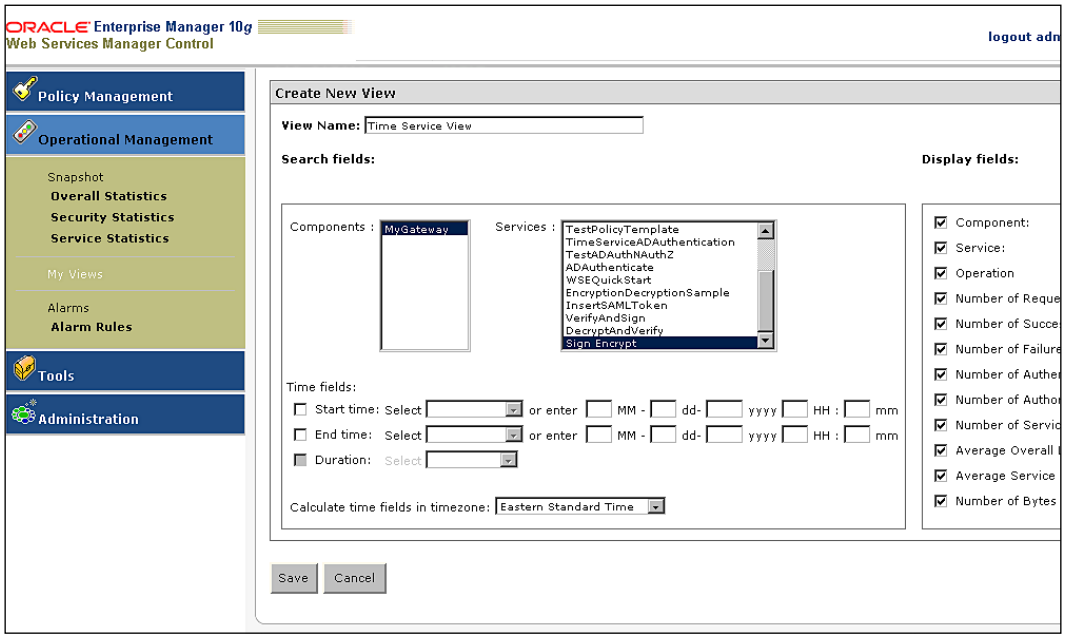
While Oracle WSM has various statistics built in and one can navigate to find out overall statistics or security statistics, it also has an option to create custom views. In the next the section, we will explore how to create custom these.

Oracle WSM Custom Views

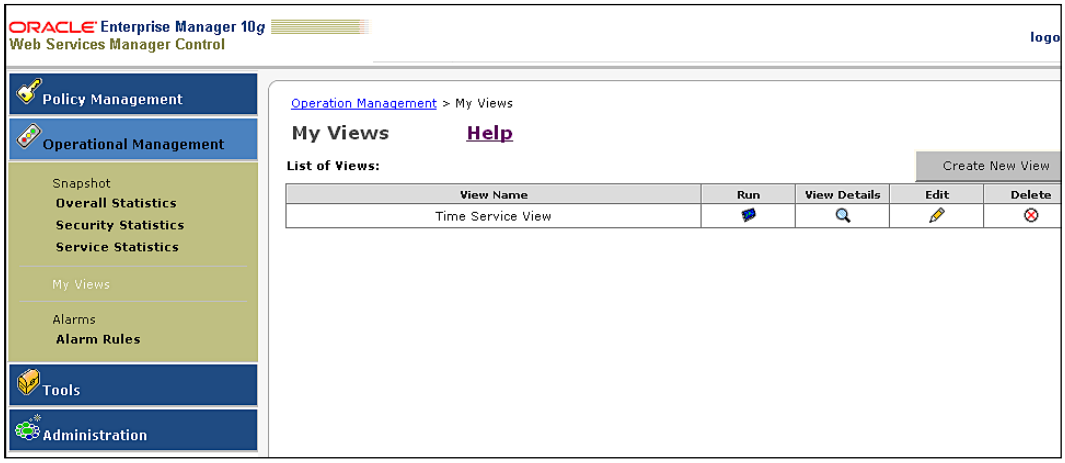
Oracle Web Services Manager maintains the statistics of all the services, and the snapshot view can help give a complete overview of all the services deployed in an organization. However, it is tedious to go through multiple levels to find out details of a particular service or a set of services that an administrator or developer may be interested in. Oracle Web Services Manager has an option to create custom views where an administrator can select only a set of services, either critical for business or the one he is responsible for, and can monitor the activities of those services.

One can create a new view from the **My Views** section under **Operational Management**, as shown in the following screen capture.

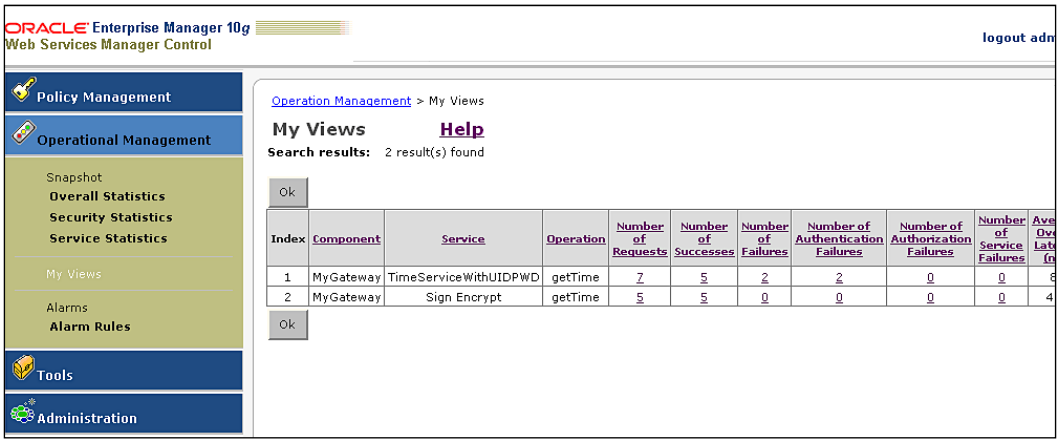
Once you click on the **Create New View** button, you can then create a new view that only has a limited number of services you are interested in. You can also select what statistics data you would be interested in and the time interval. The following screenshot shows the **Create New View** screen.



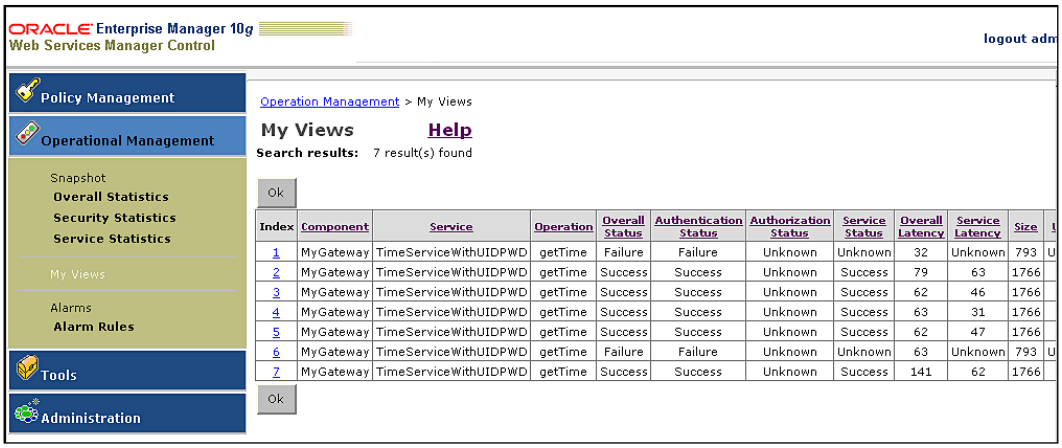
Once the services are selected and appropriate statistics data that you are interested in are selected, you can click **Save**. Once the new view is created, it will show up in the My Views section as shown in the following screenshot.



You can now click on the **Run** button to actually run the custom view to get the statistics as shown in the following screenshot.

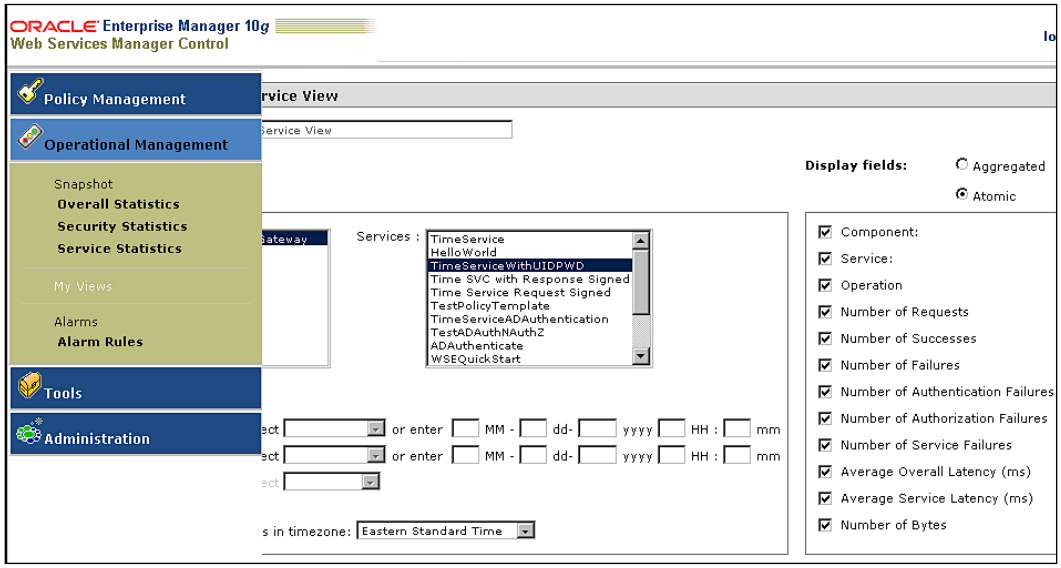


In the screenshot, you can actually see all the statistics data, such as the **Number of Requests**, successes, failures, authentication failures, authorization failures, service failures, etc. You can actually click on the hyperlinks to get the individual details, such as **Success**, **Failure**, latency of each service invocation, as shown in the following screenshot.

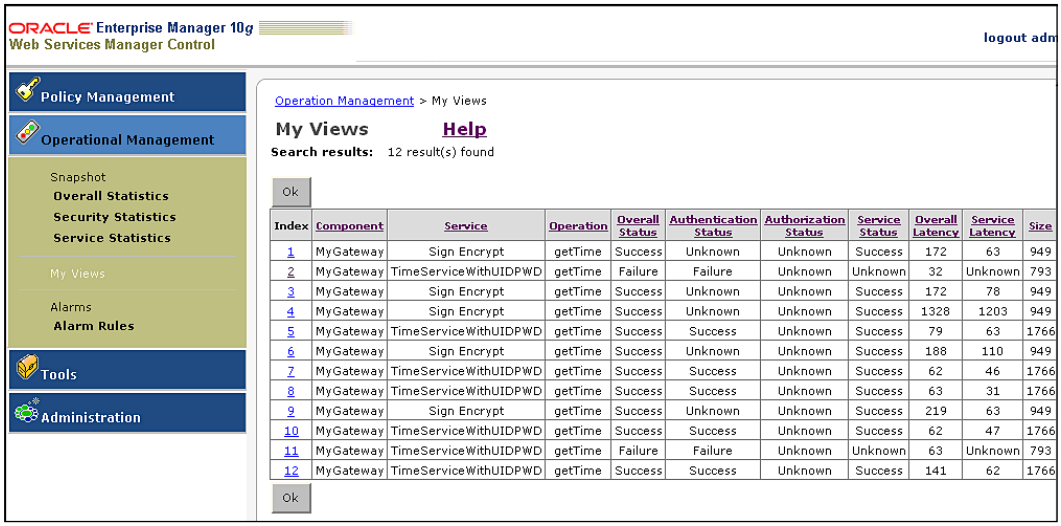


From the this screen, you can actually navigate to the message log to see the details of the message or even troubleshoot reasons for failure.

The view is created with the option called "Aggregated" which will give the total number of requests, successes, failures, etc. for each service, and you can click on each service to find out individual details about latency, success, failure, etc. However, you can also get to the details by creating the view with the option of **Atomic** as shown in the following screen capture.



In the screenshot, the **Atomic** view is selected. Once you save and run the view, you will get all the details of the individual services directly as shown in the following screen capture (as opposed to an aggregated view).

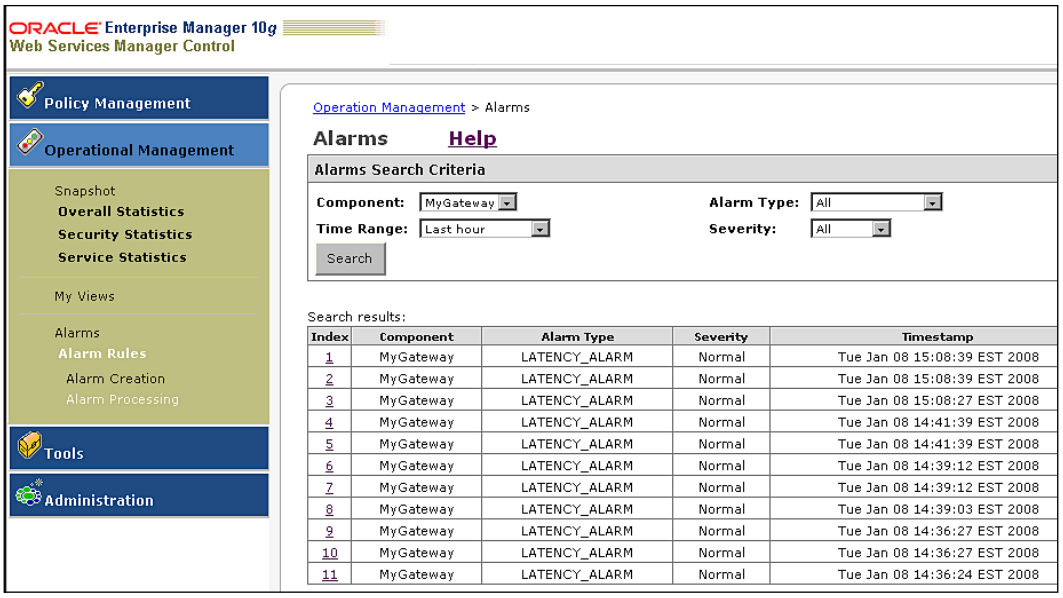


The custom view comes in handy when you want to monitor the statistics of a few services that you are interested in. However, this still gives all the information and you have to identify if there was any latency variation or security failure. Another way is to capture the details when there are any violations such as latency greater than 30, or security failures. In the next section, we will discuss alarms in detail.

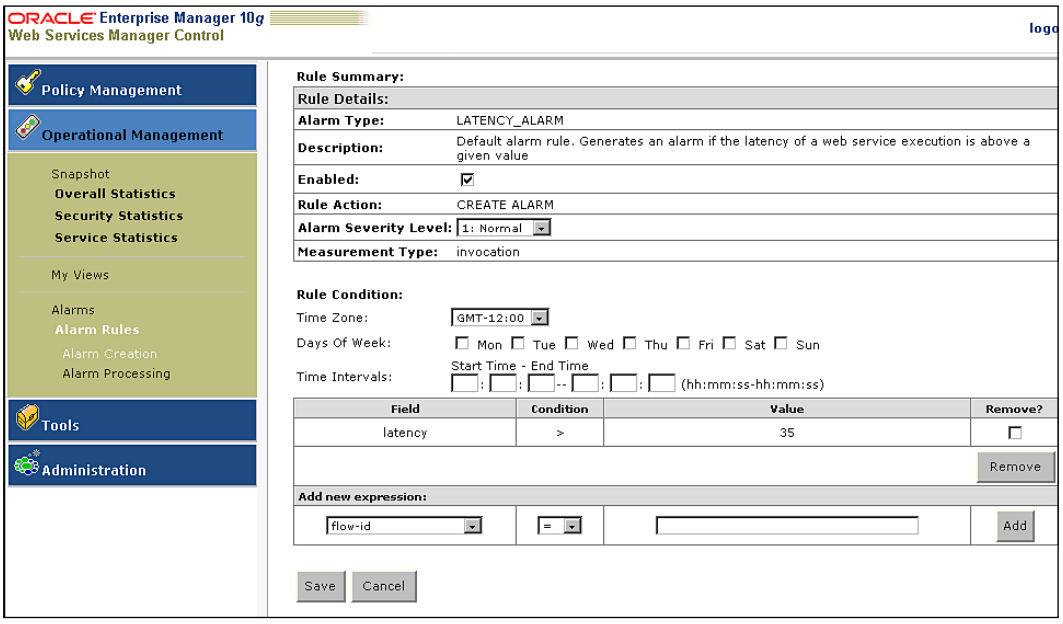
Oracle WSM Alarms

Alarms are another alternative way to identify any service failure or any other type of issue. Instead of looking through the snapshot view or custom view and then looking through all the messages or services, you can actually get notified only when there is an issue such as latency variation, authentication failure, etc.

Administrators can actually define various types of alarms. When the conditions are met, the alarm is raised. You can then click on **Alarms** from **Operational Management** to find out what alarms were created in the last hour as shown in the following screenshot.



The screenshot describes all the messages that triggered the latency alarm. In order to view the alarms, one has to actually define an alarm by clicking on **Alarm Creation**. Once you click on **Alarm Creation**, you can enter the details along with **Measurement Type** as shown in the following screen capture.



In the screenshot, the latency alarm will be triggered when the latency is greater than 35. You will be able to view the triggered alarms from the **Alarms** menu. You can also configure **Alarm Processing**, where you can trigger an external action such as email or other notification when an alarm is triggered. One of the options for the action is the ability to raise the SNMP events. When you raise SNMP events you can actually integrate the monitoring of Oracle WSM with other products such as Microsoft Operations Manager, HP Open View, etc.

Summary

In this chapter, we were able to discuss the various statistics that Oracle WSM captures and the ways in which those statistics are presented. It is important to define the service level agreement and monitor if any service violates that SLA. Also, it is important to monitor the security statistics to prevent any unauthorized access. For a larger deployment, the alarms and alarm processing comes in very handy so that the administrators can be notified when there is a problem. Oracle Web Services Manager provides the statistics to proactively monitor the service operations and the administrators and SOA architects should leverage that in their applications.

10
XML Encryption

XML encryption is a W3C standard that is now part of WS-security specification (with certain extensions for WS-security) which addresses the interoperability concerns in encryption and decryption. In Chapter 5, we discussed how Oracle Web Services Manager can encrypt and decrypt web service messages. While it is easy to configure Oracle WSM to encrypt and decrypt messages, it will be helpful if one can understand the internal details of how the encrypted XML message is formatted. Understanding the internals of XML Encryption in WS-security will help you relate to certain parameters that you select in the Oracle WSM policy steps to perform encryption or decryption. In web services, the encrypted XML is represented in an interoperable standard format, such as XML encryption. In this chapter, we will take a closer look at XML encryption standard from W3C.

XML Encryption and Web Services

Confidentiality in web services is ensured by encrypting the confidential data in the web service messages. Encryption is nothing but a process of applying an algorithm using an encryption key over the plain text to create the cipher text (encrypted data). While this sounds simple, applications need to know the algorithm used to encrypt the data, how the encrypted data is represented in the final output, etc. Traditional applications tend to encrypt the entire message even if there is only one piece of information that is confidential. They also require both the sender and the recipient to use the same software, such as PGP (Pretty Good Privacy), for encryption and decryption.

In the web services world, it is not practical to force everyone to use one set of programs to encrypt or decrypt messages, and it may not be necessary to encrypt the entire message every time a piece of confidential data needs to be protected (encrypted). The ideal solution would be to:

- Encrypt only portions of the XML that needs to be encrypted.
- Service provider and consumer should not be tied to the same set of programs to encrypt and decrypt.

In order for anyone who has the appropriate keys to decrypt the data, the encrypted data should be represented in such a way that the application can determine:

- Where the encrypted data is placed inside the XML message.
- What encryption algorithm is used to encrypt the data.
- Any other cryptographic parameters.

The said requirements are addressed by the XML encryption specification from W3C where the XML encryption schema describes how the encrypted data can be represented. We will take a closer look at XML encryption in the next section.

XML Encryption Schema

By definition, the XML encryption schema is nothing but an XML schema that outlines how the encrypted data can be described in an XML format. Any application that can understand the XML encryption schema should be able to encrypt and display data as per the schema, and should also be able to decrypt the data. The following example XML represents encrypted data in the XML format.

```
    <xenc:EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" Type="http://www.
w3.org/2001/04/xmlenc#Element" Id="ED">
      <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/
xmlenc#tripledes-cbc"/>
      <dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
        <dsig:KeyName>EncryptedElementWithSymmKey</dsig:KeyName>
      </dsig:KeyInfo>
      <xenc:CipherData>        <xenc:CipherValue>nrNckdqE5O1CCZkjA0RM9
s7mOYv8t4QUJ/tE1ONdfUe8zWaXeJuDYAl5dQ4Ypfpu</xenc:CipherValue>
      </xenc:CipherData>
    </xenc:EncryptedData>
```

In the example, the encrypted data is represented by the `CipherValue` XML element and the algorithm used, which is **Triple DES,** represented by `EncryptionMethod` XML element. `EncryptedData` XML element represents all the information together such as `EncryptionMethod` to describe the algorithm, `KeyInfo` to describe the symmetric key information, `CipherData` to represent the cipher text or encrypted text.

This example describes a symmetric key encryption scenario where both the sender and recipient are aware of the encryption key (communicated over phone or other means). The asymmetric key encryption addresses the key exchange problem by encrypting the data using the public key of the recipient, and the recipient can decrypt the data using the private key, known only to the recipient.

Encrypting the data using a public key is usually not recommended for high volume transactions as it is CPU intensive. Instead, a combination of symmetric and public key methods are used, as follows:

- Symmetric key is generated at random.
- Plain text data is encrypted using the symmetric key.
- Symmetric key (also called encryption key) is then encrypted using the public key of recipient.

The mentioned process can be easily translated to the XML encryption schema by means of the `EncryptedKey` XML element. The following example XML message describes how the EncryptedKey is represented.

```
<xenc:EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#"
   xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
   Type="http://www.w3.org/2001/04/xmlenc#Content" Id="Lock">
<xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/
xmlenc#tripledes-cbc"/>
<dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
<xenc:EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#"
   xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" Id="EK">
<xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/
xmlenc#rsa-1_5"/>
<dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
  <dsig:X509Data>
    <dsig:X509Certificate>MIICajCCAdMCBD3RhNAwDQYJKoZIhvcNA</dsig:
X509Certificate>
  </dsig:X509Data>
  </dsig:KeyInfo>
<xenc:CipherData>
   <xenc:CipherValue>WIRgk=</xenc:CipherValue>
```

```
</xenc:CipherData>
<xenc:ReferenceList>
   <dsig:DataReference URI="#Lock"/>
</xenc:ReferenceList>
</xenc:EncryptedKey>
</dsig:KeyInfo>
<xenc:CipherData>
<xenc:CipherValue>/cpa26ZsDr1AF8</xenc:CipherValue>
</xenc:CipherData>
</xenc:EncryptedData>
```

This XML message is a very good example of how flexible the XML encryption schema is. From the two examples shown, it is clear that the XML encryption schema is comprised of two key XML elements, `EncryptedData` and `EncryptedKey`, which we will explore in detail in the following sections.

EncryptedData

`EncryptedData` element represents the encrypted data information so that the recipient can actually decrypt the data. The following example shows the `EncryptedData` element that describes the information about the encrypted data.

```
    <xenc:EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" Type="http://www.
w3.org/2001/04/xmlenc#Element" Id="ED">
      <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/
xmlenc#tripledes-cbc"/>
      <dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
        <dsig:KeyName>EncryptedElementWithSymmKey</dsig:KeyName>
      </dsig:KeyInfo>
      <xenc:CipherData>        <xenc:CipherValue>nrNckdqE5O1CCZkjA0RM9
s7mOYv8t4QUJ/tE1ONdfUe8zWaXeJuDYAl5dQ4Ypfpu</xenc:CipherValue>
      </xenc:CipherData>
    </xenc:EncryptedData>
```

In this example, `EncryptedData` element consists of various key information required to decrypt the data, such as:

- `EncryptionMethod`: XML element that describes that the algorithm is Triple DES.
- `KeyInfo`: XML element that describes the `KeyName` so that the recipient can use it to derive the actual symmetric key.
- `CipherData`: XML element that contains the `CipherValue` element, which is the actual encrypted data.

The `EncryptedData` element is actually derived from `EncryptedType` XML element and the structure of `EncryptedType` (or its instance `EncryptedData`) is:

```
<complexType name='EncryptedType' abstract='true'>
  <sequence>
  <element name='EncryptionMethod' type='xenc:EncryptionMethodType'
minOccurs='0'/>
  <element ref='ds:KeyInfo' minOccurs='0'/>
  <element ref='xenc:CipherData'/>
  <element ref='xenc:EncryptionProperties' minOccurs='0'/>
  </sequence>
  <attribute name='Id' type='ID' use='optional'/>
  <attribute name='Type' type='anyURI' use='optional'/>
  <attribute name='MimeType' type='string' use='optional'/>
  <attribute name='Encoding' type='anyURI' use='optional'/>
</complexType>
```

The `EncryptedType` schema shown describes:

- `Id` attribute: Uniquely identifies the `EncryptedData` or `EncryptedKey` element in the document.
- `Type`, `MimeType` and `Encoding` attributes: These optional attributes describe the encrypted text. In our example, the plain text that was encrypted was an XML element. It is represented by `Type = http://www.w3.org/2001/04/xmlenc#Element`.
- `EncryptionMethod` element: This describes the algorithm used to encrypt the data. This element is of a type called `EncryptionMethodType` which is described later.
- `CipherData` element: This describes the encrypted text, i.e. the cipher text.
- `KeyInfo` element: This describes the encryption key used to encrypt the data.

EncryptionMethodType

The `EncryptionMethod` element described in the `EncryptedType` schema is of a data type called `EncryptionMethodType`. The `EncryptionMethodType` data type describes the information about the encryption method, such as algorithm and any other additional parameters based on the type, algorithm.

The `EncryptionMethodType` element should be generic enough to support both symmetric and asymmetric algorithms, and the schema is described as:

```
<complexType name='EncryptionMethodType' mixed='true'>
  <sequence>
    <element name='KeySize' minOccurs='0' type='xenc:KeySizeType'/>
    <element name='OAEPparams' minOccurs='0' type='base64Binary'/>
```

```
        <any namespace='##other' minOccurs='0' maxOccurs='unbounded'/>
      </sequence>
      <attribute name='Algorithm' type='anyURI' use='required'/>
    </complexType>
```

EncryptionMethodType Schema

The `Algorithm` attribute in the schema shown describes the encryption algorithm used. In our example, the algorithm used is Triple DES as shown in the following code..

```
<xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/
xmlenc#tripledes-cbc"/>
```

Besides the `Algorithm` attribute, the `EncryptionMethod` can also have child elements such as `KeySize`, OAEP parameters, and any other additional elements. The OAEP parameters and any other XML elements are based on the value of the `Algorithm URI` attribute. For instance, the presence of `KeySize` element would make the schema invalid if the `Algorithm` attribute did not permit the `KeySize` element.

CipherData Element

The `CipherData` element is the one that actually describes the information regarding the cipher text. The typical encryption process is nothing but:

```
Plain Text + Encryption Algorithm = Cipher Text.
```

The cipher text is required in order to decrypt to obtain the plain text information. In the XML world, the `CipherData` element represents the information about cipher text, and the schema is described as:

```
<element name='CipherData' type='xenc:CipherDataType'/>
  <complexType name='CipherDataType'>
     <choice>
       <element name='CipherValue' type='base64Binary'/>
       <element ref='xenc:CipherReference'/>
     </choice>
   </complexType>
```

As you can see from the schema, the `CipherData` element can either contain the `CipherValue` element or a reference to the cipher value by means of `CipherReference`. (Note: the `choice` element is an XML syntax to make a selection among the list of choices.)

`CipherValue` element describes the encrypted text in base64 binary and the `CipherReference` element is a reference to URI where the cipher text can be found. In our `EncryptedData` example, the cipher text is represented by `CipherValue` element in **base64** binary format as shown in the following code.

```
<xenc:CipherData>        <xenc:CipherValue>nrNckdqE5O1CCZkjA0RM9s7mOYv
8t4QUJ/tE1ONdfUe8zWaXeJuDYAl5dQ4Ypfpu</xenc:CipherValue>
      </xenc:CipherData>
```

EncryptedKey Element

The `EncryptedData` element describes information about the cipher text, algorithm and the encryption key. When the encryption key is a symmetric key, both the sender and recipient should share the encryption key in an out of band mechanism. It will become increasingly difficult to share the encryption key when there are too many consumers accessing a web service that contains the encrypted information. It will also be a security risk to share the same encryption key across all the consumers. One possible way to address the key distribution is to encrypt the symmetric encryption key and the process would be:

- Symmetric key is generated at run time.
- Plain text is encrypted using the symmetric encryption key.
- Encryption key is then encrypted using the public key of the recipient.
- Encrypted encryption key is distributed along with the XML message (i.e. EncryptedKey).

Like `EncryptedData` element, the `EncryptedKey` element is also derived from the `EncryptedType` element. The `EncryptedKey` element represents the information about the encryption key in an encrypted format, along with additional information that is required to decrypt the `EncryptedData` element using the encryption key. The `EncryptedKey` element schema is described as:

```
<element name='EncryptedKey' type='xenc:EncryptedKeyType'/>
<complexType name='EncryptedKeyType'>
  <complexContent>
    <extension base='xenc:EncryptedType'>
      <sequence>
        <element ref='xenc:ReferenceList' minOccurs='0'/>
        <element name='CarriedKeyName' type='string' minOccurs='0'/>
      </sequence>
      <attribute name='Recipient' type='string' use='optional'/>
    </extension>
  </complexContent>
</complexType>
```

The EncryptedKey schema has the same elements and attributes as the EncryptedData element, with additional information such as ReferenceList, CarriedKeyName and Recipient.

The optional ReferenceList element can contain a list of references to data and keys that are encrypted using the enclosed encryption key. The CarriedKeyName and Recipient are also optional elements that can carry a friendly key name and any specific hint about the encryption key respectively.

KeyInfo Element

In order to decrypt the EncryptedData or EncryptedKey element, the recipient has to know about the encryption key. The recipient may already know about the encryption key, in which case there may not be any need to exchange the KeyInfo element. However, when the recipient does not know about the encryption key, the KeyInfo element can provide the necessary information so that recipient can decrypt the message.

In the case of symmetric key encryption, the recipient should know something about the encryption key in order to decrypt the EncryptedData element. In this case, the sender can attach a friendly KeyName as a part of the KeyInfo element as shown in the following code:

```
<dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
  <dsig:KeyName>TestKeyname</dsig:KeyName>
</dsig:KeyInfo>
```

In case of asymmetric key encryption, the sender can send information about the X509 certificate. The KeyInfo element schema is described as:

```
<element name="KeyInfo" type="ds:KeyInfoType"/>
<complexType name="KeyInfoType" mixed="true">
  <choice maxOccurs="unbounded">
    <element ref="ds:KeyName"/>
    <element ref="ds:KeyValue"/>
    <element ref="ds:RetrievalMethod"/>
    <element ref="ds:X509Data"/>
    <element ref="ds:PGPData"/>
    <element ref="ds:SPKIData"/>
    <element ref="ds:MgmtData"/>
    <any processContents="lax" namespace="##other"/>
    <!-- (1,1) elements from (0,unbounded) namespaces -->
  </choice>
  <attribute name="Id" type="ID" use="optional"/>
</complexType>
```

Summary

The XML encryption schema addresses the interoperability issues in exchanging encrypted data and it adds a lot of value as a part of the WS-security specification. In the web services world, the confidential data is protected by encrypting the data and representing, as per XML encryption schema. In this chapter, we took a closer look at XML encryption schema, how the different elements are organized, represented and how they are linked together.

11
XML Signature

In any data exchange, the integrity of the message is very important—especially the ones that involve business critical data where any data changed by an intruder can affect the day-to-day business. One way to ensure that no one else tampered with the message (integrity) is to digitally sign the message. While digital signatures are not new, traditionally applications used tools such as PGP (pretty good privacy) to sign and verify messages. The drawback of PGP, or any other tool, is that it requires both the sender and recipient to use the same tool to sign and verify messages, which is a bottleneck in the web services world. In the web services world, the sender and recipient should be able to use any software tool and any algorithm they like to generate and verify messages, as long as the signed messages are represented in an industry standard way. XML signature is an interoperable industry standard from W3C that addresses how the digitally-signed messages are represented or described in an XML format. Oracle WSM can digitally sign and verify the web service messages. In this chapter, we will take a closer look at XML signature specification from W3C.

XML Signature and Web Services

The integrity of data can only be ensured by creating a digest of the message and then encrypting the digest value, i.e. digitally signing the message. Now in order for the recipient to validate the signature, the recipient has to know:

- What parts of the message were digitally signed.
- What digest algorithm was used.
- Where the encrypted digest value is.
- What algorithm was used to encrypt the digest value.

In the web services world, it is not feasible to communicate where the digest values are described in the XML message, what algorithm is used to create the digest, etc. All the information that is required for the recipient to successfully validate the digital signature should be described in such a way that any recipient is able to validate the signature.

The XML signature specification from W3C addresses how the digital signature information is represented in an extensible mark up language format so that the recipient can understand:

- What parts of the XML message are digitally signed.
- What digest algorithm was used to create the digest value.
- What algorithm was used to digitally sign the message.
- Where is the key information is located within the XML message.

The XML signature schema describes the syntax about how the digital signature information should be described in an XML document and how it fits within the WS-security framework to ensure the integrity of messages. In the next section, we will describe in detail the XML signature schema and how it is used in WS-security to represent the signature.

XML Signature Schema

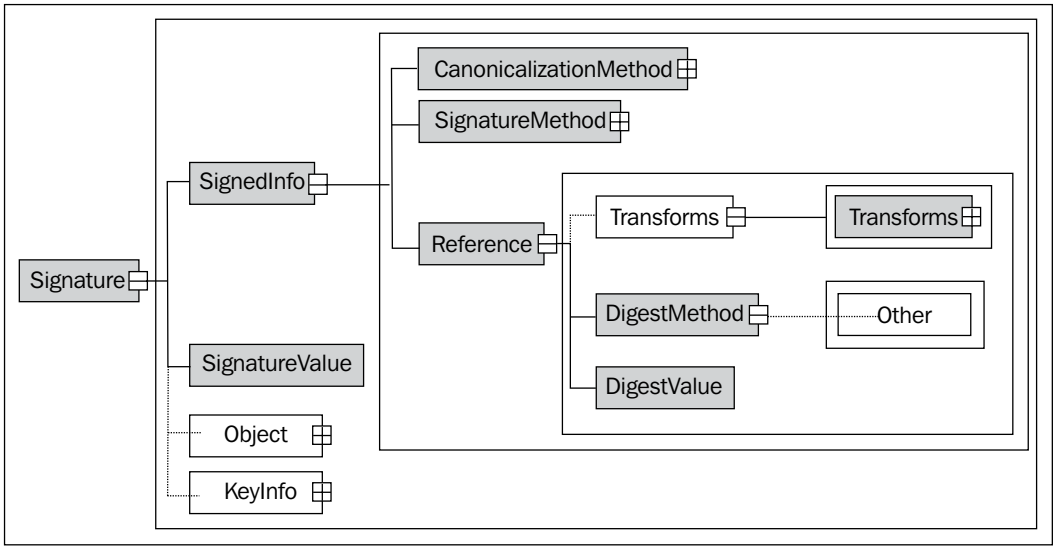
The XML signature schema is a W3C standard that describes how the digital signature information is represented in an XML document. XML signature schema can not only describe the digital signature of an XML element or XML element content, but also an image or an externally-referenced entity. In any case, the recipient of the XML signature message has to know:

- Data that is being signed.
- Digest value of that data.
- Digest algorithm used.
- Digital signature information, such as signature value, key information.

The digitally-signed message that adheres to the XML signature schema always starts with a **signature** XML element which contains the following main child elements:

- SignedInfo
- SignatureValue
- KeyInfo
- Object

The mentioned XML elements can have there own child elements that can specify additional instructions to process the digitally-signed message. The following figure shows an overview of the XML signature schema with the signature element as the root element and its child elements. The figure also shows the order in which those elements appear, which is very important, as described in the XML signature schema.



When the digital signature is applied on the web service message, the signature element is inserted, which contains the reference to the data that is signed along with digest value, signature value and key information, as shown by the following code.

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope soap:encodingStyle="http://schemas.xmlsoap.org/
soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:
soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:
soapenc="http://schemas.xmlsoap.org/soap/encoding/">
   <soap:Header>
         <wsse:Security xmlns:wsse="http://docs.oasis-open.
org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd" soap:mustUnderstand="1">
               <wsse:BinarySecurityToken ValueType="http://docs.
```

```
oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-
1.0#X509v3" EncodingType="http://docs.oasis-open.org/wss/2004/01/
oasis-200401-wss-soap-message-security-1.0#Base64Binary" wsu:
Id="_VLL9yEsi09I9f5ihwae2lQ22" xmlns:wsu="http://docs.oasis-open.
org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">Secur
ityTOkenoKE2ZA==</wsse:BinarySecurityToken>
                <dsig:Signature xmlns="http://www.w3.org/2000/09/
xmldsig#" xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
                        <dsig:SignedInfo>
                                <dsig:CanonicalizationMethod
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                                <dsig:SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
                                <dsig:Reference URI="#ishUwYWW2AAthrx
hlpv1CA22">
                                        <dsig:Transforms>
                                                <dsig:Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                                        </dsig:Transforms>
                                        <dsig:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                                        <dsig:DigestValue>ynuqANuYM3qzh
dTnGOLT7SMxWHY=</dsig:DigestValue>
                                </dsig:Reference>
                                <dsig:Reference URI="#UljvWiL8yjedImz
6zy0pHQ22">
                                        <dsig:Transforms>
                                                <dsig:Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                                        </dsig:Transforms>
                                        <dsig:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                                        <dsig:DigestValue>9ZebvrbVYLiPZ
v1BaVLDaLJVhwo=</dsig:DigestValue>
                                </dsig:Reference>
                        </dsig:SignedInfo>
                        <dsig:SignatureValue>QqmUUZDLNeLpAEFXndiBLk=
</dsig:SignatureValue>
                        <dsig:KeyInfo>
                                <wsse:SecurityTokenReference
xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd" wsu:Id="_7vjdWs1ABULkiLeE7Y4lAg22"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-wssecurity-utility-1.0.xsd">
```

```
                                        <wsse:Reference URI="#_
VLL9yEsi09I9f5ihwae2lQ22"/>
                                    </wsse:SecurityTokenReference>
                            </dsig:KeyInfo>
                        </dsig:Signature>
                        <wsu:Timestamp xmlns:wsu="http://docs.oasis-open.
org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd" wsu:Id="UljvWiL8yjedImz6zy0pHQ22">
                            <wsu:Created>2007-11-16T15:13:48Z</wsu:
Created>
                        </wsu:Timestamp>
                </wsse:Security>
        </soap:Header>
        <soap:Body wsu:Id="ishUwYWW2AAthrxhlpv1CA22" xmlns:wsu="http://
docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd">
                <n:getTimeResponse xmlns:n="urn:Test:GetTime">
                        <Result xsi:type="xsd:string">10:13 AM</Result>
                </n:getTimeResponse>
        </soap:Body>
</soap:Envelope>
```

The above example is taken from Chapter 6 where the web service response was digitally signed. In this chapter, we will be talking a closer look at the `Signature` element. In order to understand how the `Signature` element was formed and how the options that you select in the Oracle WSM signature policy step can affect the signed message, we will take a closer look at the XML signature schema elements in detail.

Signature Element

The `Signature` element is the root XML element, as per the XML signature schema that contains information such as:

- **SignedInfo**—Information about which data elements were signed and the transformations that were applied before the digest value was generated.
- **SignatureValue**—Base 64 encoded digital signature value.
- **KeyInfo**—Optional element that describes the key that was used to sign the message.
- **Object**—Optional element which can embed the actual data object itself, if required.
- **Id**—`Id` attribute that uniquely identifies the signature element inside the XML document.

The **Signature** element schema can be described as:

```
<element name="Signature" type="ds:SignatureType"/>
<complexType name="SignatureType">
  <sequence>
    <element ref="ds:SignedInfo"/>
    <element ref="ds:SignatureValue"/>
    <element ref="ds:KeyInfo" minOccurs="0"/>
    <element ref="ds:Object" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="Id" type="ID" use="optional"/>
</complexType>
```

SignedInfo Element

`SignedInfo` element contains the maximum information about the digital signature such as the reference to the data being signed, list of transformations applied to the data, digest value, etc. The `SignedInfo` element schema is described as:

```
<element name="SignedInfo" type="ds:SignedInfoType"/>
   <complexType name="SignedInfoType">
     <sequence>
       <element ref="ds:CanonicalizationMethod"/>
       <element ref="ds:SignatureMethod"/>
       <element ref="ds:Reference" maxOccurs="unbounded"/>
     </sequence>
     <attribute name="Id" type="ID" use="optional"/>
   </complexType>
```

The `SignedInfo` element contains:

- **CanonicalizationMethod**: Element that describes the canonicalization algorithm that is applied. There are actually four types of canonicalization algorithms—C14n, C14n with comments, exclusive C14n, and exclusive C14n with comments.
- **SignatureMethod**: It describes the signature algorithm used, along with any additional information about the algorithm.
- **Reference**: The `Reference` element holds references to the data element and its associated digest value.

Reference Element

The `Reference` element that is a part of the `SignedInfo` element contains the digest value of the data being signed along with a list of transformations that were applied to the data. The `Reference` element schema is described as:

```
<element name="Reference" type="ds:ReferenceType"/>
   <complexType name="ReferenceType">
     <sequence>
       <element ref="ds:Transforms" minOccurs="0"/>
       <element ref="ds:DigestMethod"/>
       <element ref="ds:DigestValue"/>
     </sequence>
     <attribute name="Id" type="ID" use="optional"/>
     <attribute name="URI" type="anyURI" use="optional"/>
     <attribute name="Type" type="anyURI" use="optional"/>
   </complexType>
```

The `Reference` element typically describes the digest value and the digest method that is used to create the digest value. It also describes the data upon which the digest algorithm is applied and the list of transformations that were applied. In the `Reference` element schema:

- `URI` attribute describes the reference to the data that is being digitally signed.
- `Transforms` element describes the list of transformations that are applied to the data. One can apply XSL transformation or any other transformation to derive the actual data from the referenced URI.
- `DigestMethod` element describes the algorithm used to create the digest value.
- `DigestValue` element is the Base 64 encoded digest value.

The following example shows the `Reference` element, which is a part of the web service response message that was digitally signed.

```
                <dsig:Reference URI="#UljvWiL8yjedImz6zy0pHQ22">
                                  <dsig:Transforms>
                                        <dsig:Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                                  </dsig:Transforms>
                                  <dsig:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                                  <dsig:DigestValue>9ZebvrbVYLiPZ
v1BaVLDaLJVhwo=</dsig:DigestValue>
                            </dsig:Reference>
                      </dsig:SignedInfo>
```

In the example, the URI value `UljvWiL8yjedImz6zy0pHQ22` points to the data within the web service message. The `Transforms` element contains `Transform` element that describes the exclusive C14 algorithm as described by the following code.

```
                                    <dsig:Transforms>
                                          <dsig:Transform
  Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                                    </dsig:Transforms>
```

What this means is that exclusive canonical transformation is applied to the data referenced by the URI, and then the digest algorithm is applied. The `DigestMethod` shows that it is a SHA1 algorithm and the actual digest value is described in the `DigestValue` element.

Transforms Element

The `Transforms` element contains a list of transformations that are applied to the data before the digest value is calculated. The Transforms schema is described as:

```
<element name="Transforms" type="ds:TransformsType"/>
<complexType name="TransformsType">
  <sequence>
    <element ref="ds:Transform" maxOccurs="unbounded"/>
  </sequence>
</complexType>
<element name="Transform" type="ds:TransformType"/>
<complexType name="TransformType" mixed="true">
  <choice minOccurs="0" maxOccurs="unbounded">
    <any namespace="##other" processContents="lax"/>
    <!-- (1,1) elements from (0,unbounded) namespaces -->
    <element name="XPath" type="string"/>
  </choice>
  <attribute name="Algorithm" type="anyURI" use="required"/>
</complexType>
```

The digest value is unique on the data that is applied. So when the recipient has to validate the signature, the digest algorithm should be applied over the same data to get the exact digest value.

Consider, for example, that the web service response (or request) message can have a lot of information and there may be only a portion of the XML message that is critical for the business to ensure the message integrity. In this case, either XSL transformation or Xpath can be applied to select only a portion of the message.

KeyInfo Element

In order to verify the digital signature, the recipient should have access to the sender's public key. The sender can either send the actual public key itself as a part the XML signature message, or can send enough information to derive the key. In either case, web services that contain the digitally-signed message should contain information about the sender's public key. The `KeyInfo` element schema is described as:

```
<element name="KeyInfo" type="ds:KeyInfoType"/>
<complexType name="KeyInfoType" mixed="true">
  <choice maxOccurs="unbounded">
    <element ref="ds:KeyName"/>
    <element ref="ds:KeyValue"/>
    <element ref="ds:RetrievalMethod"/>
    <element ref="ds:X509Data"/>
    <element ref="ds:PGPData"/>
    <element ref="ds:SPKIData"/>
    <element ref="ds:MgmtData"/>
    <any processContents="lax" namespace="##other"/>
    <!-- (1,1) elements from (0,unbounded) namespaces -->
  </choice>
  <attribute name="Id" type="ID" use="optional"/>
</complexType>
```

It is very clear from the `KeyInfo` element schema that the X509 certificate data itself can be sent as a part of the `X509Data` element or just a reference to the certificate can be sent as a part of the `KeyName` element.

Summary

The XML signature specification from W3C addresses the interoperability challenges in exchanging the digital signature information between two different applications. In this chapter, we looked at different components of the XML signature and how they are related to each other in forming a meaningful XML signature message. XML signature is widely used in web services security as a mechanism to ensure the integrity of the message. In the previous chapter, we discussed XML encryption and the interoperable standard to exchange the encrypted information. Both XML encryption and XML signature play a significant part in web services security, either individually or in combination. In the next chapter, we will discuss in detail how to sign and encrypt the web service message in one transaction.

12
Sign and Encrypt

In the earlier chapters, we learned how Oracle Web Services Manager can be leveraged to digitally sign and encrypt messages to ensure the integrity and confidentiality of the data. But at times it is necessary to sign and encrypt the same message and in this chapter, we will discuss how to sign and encrypt using Oracle Web Services Manager.

Overview of Sign and Encrypt

Let's consider the example of a web service which accepts credit card details to charge to that credit card and that is being consumed by an online electronics web site. In this type of transaction it is important to ensure that the:

- Credit card number, expiration date, amount, etc, were sent from an authorized consumer, i.e. electronics web site, and the data was not modified by any one other than the electronics company.
- Credit card details are kept confidential over the wire.

In order to ensure that the credit card data was not modified by anyone other than the electronics company web site, the message should be digitally signed, and in order to ensure the confidentiality of the credit card details, the message should be encrypted. While we have discussed how to digitally sign and encrypt the message, we have not explored which one should happen first.

While signature and encryption can happen in any order, it is important to understand the impact of the order in which the signature and encryption are applied. In our example, we can either encrypt the credit card details and then sign the encrypted data or sign the credit card details and then encrypt the data.

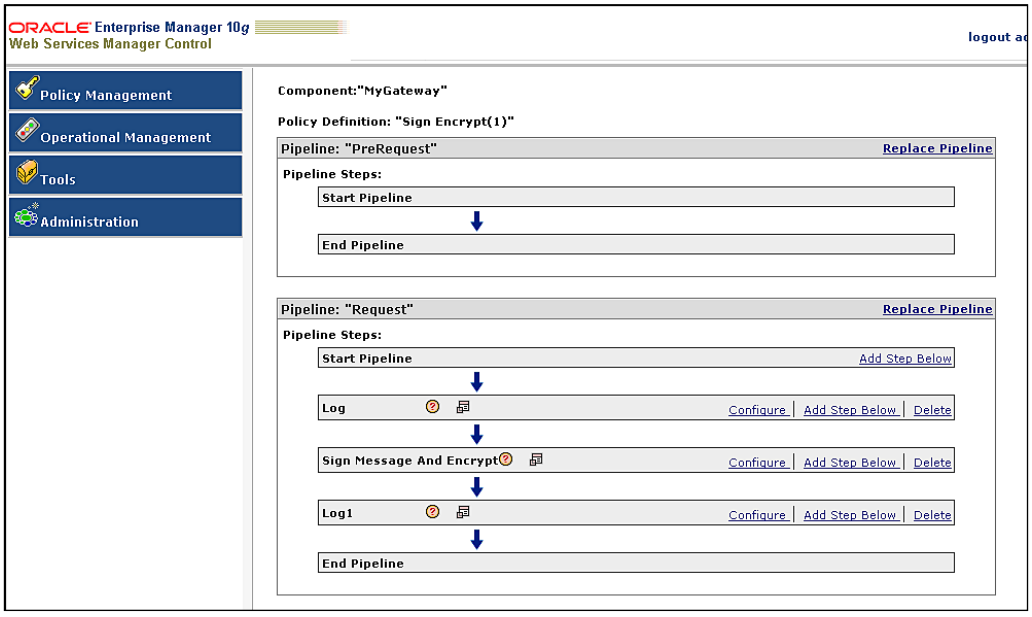
When the credit card information is encrypted first and the encrypted data is then digitally signed, the digital signature is applied over the encrypted data, not on the actual credit card details. In other words, the signature only ensures that the encrypted data is not tampered with, but it does not ensure the integrity of the actual plain text, i.e. credit card details.

Now when the credit card information is signed first and then encrypted, it ensures that no one else tampered with the original credit card details and that the information is also kept confidential by means of encryption.

In Oracle Web Services Manager, there is a built-in policy step called "sign and encrypt" which will digitally sign the message first and then perform the encryption. In the next section, we will describe in detail how to configure the policy for a web service to sign and encrypt, and also a policy for web services to decrypt and verify a signature.

Signing and Encrypting Message

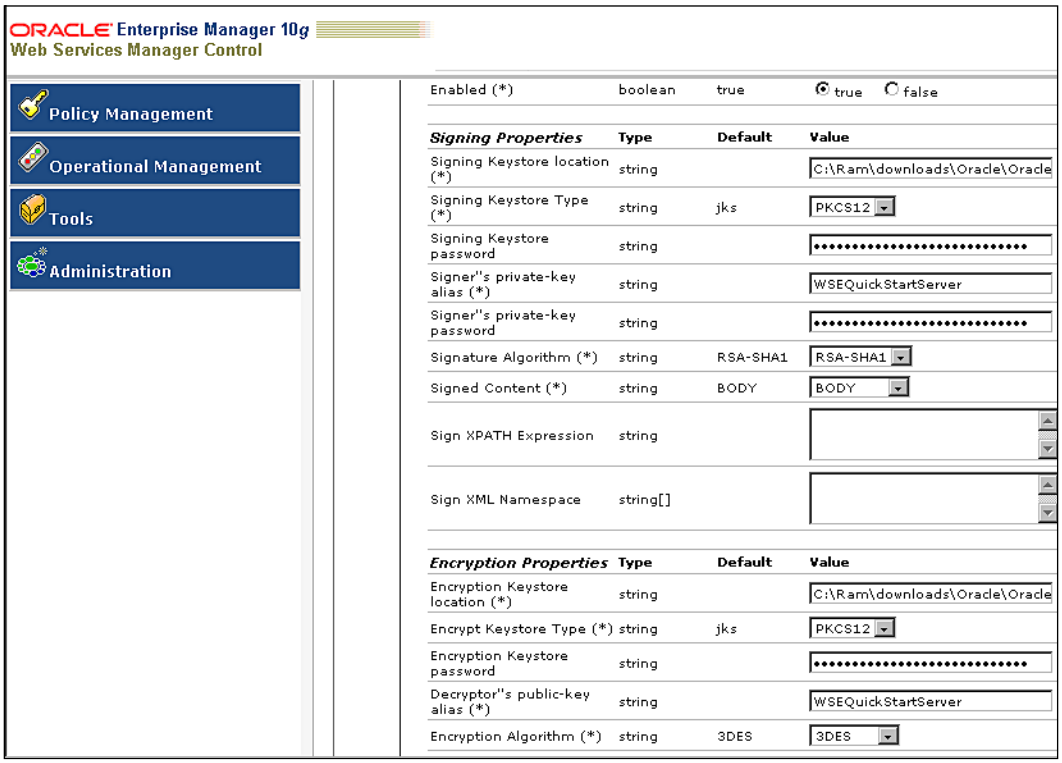
In Oracle Web Services Manager, it is easy to sign and encrypt the SOAP messages in just one policy step. Let's consider an example where you want to access an external web service, and the request should be digitally signed first and then encrypted. In this case, the sign and encrypt policy step can be added in the request pipeline so that the messages are digitally signed and then encrypted. The following screenshot shows how the **Sign Message And Encrypt** step can be added to the **Request** pipeline.



Now if you click to configure the **Sign Message And Encrypt** policy step, you have to enter two sets of information:

- One set of information to digitally sign the message
- Another set of information to encrypt the signed message

In the following screenshot, you can see how to configure the **Sign Message And Encrypt** policy step.



The **Signing Properties** information is used to digitally sign the data. In this case, you have to enter the **PKCS12** key store location and password along with the password to access the private key. A private key is required as the data will be digitally signed.

The Encryption Properties information is used to encrypt the message. In this case, you have to enter the key store location, keystore type, public key from keystore file and the algorithm used to encrypt the data.

The following steps happen within the sign and encrypt step when Oracle WSM digitally signs the message and then encrypts the data.

- The request message is first digitally signed.
    - Message is digitally hashed first.
    - Digitally signed using the sender's private key.
- The signature is then encrypted.
    - A new symmetric key is generated at run time.
    - The signature element (digital signature data) is then encrypted using the symmetric key.
    - The encrypted symmetric key is then encrypted using the public key of the recipient.
- The `EncryptedData` element is then sent along with the message.

Once Oracle WSM receives the signed and encrypted message, it can then decrypt the message and validate the signature before proceeding further, and Oracle WSM will perform the following steps:

- The SOAP message will decrypt the encrypted data first using the private key of the recipient.
- The digital signature is regenerated using the public key of the recipient.
- The signature value is compared and validated with the one that was decrypted.

In the following sample XML message, the web service request to get the time is first digitally signed and then encrypted. The SOAP body message now contains the encrypted value. But before the SOAP body was encrypted, it was digitally signed and attached to the SOAP header.

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns="urn:Test:GetTime" xmlns:xsd="http://www.w3.org/2001/
XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
   <soap:Header>
          <wsse:Security soap:mustUnderstand="1" xmlns:wsse="http://
docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-
1.0.xsd" xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-wssecurity-secext-1.0.xsd">
                <wsse:BinarySecurityToken ValueType="http://docs.
oasis-open.org/wss/2004/01/oasis-200401-wss-x509-ten-profile-
1.0#X509v3" EncodingType="http://docs.oasis-open.org/wss/2004/01/
oasis-200401-wss-soap-message-security-1.0#Base64Binary" wsu:Id="
```

```
aZ1TmJt9p7wCfugDdlGweDA22" xmlns:wsu="http://docs.oasis-open.org/
wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">oKE2ZA==</
wsse:BinarySecurityToken>
                <xenc:EncryptedKey xmlns="http://www.w3.org/2001/04/
xmlenc#" xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
                        <xenc:EncryptionMethod Algorithm="http://www.
w3.org/2001/04/xmlenc#rsa-1_5"/>
                        <dsig:KeyInfo xmlns:dsig="http://www.
w3.org/2000/09/xmldsig#">
                                <SecurityTokenReference xmlns="http://
docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-
1.0.xsd">
                                        <Reference URI="#aZ1TmJt9p7wCfug
DdlGweDA22" ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-x509-token-profile-1.0#X509v3"/>
                                </SecurityTokenReference>
                        </dsig:KeyInfo>
                        <xenc:CipherData>
                                <xenc:CipherValue>jXyDqgTC0DvyXkS7lz+Qq1
P2MMG4G5W5zy4D3K2s+S2b7o=</xenc:CipherValue>
                        </xenc:CipherData>
                        <xenc:ReferenceList>
                                <xenc:DataReference URI="#_
4LDVnkQFr3DlbkZFa91f2w22"/>
                        </xenc:ReferenceList>
                </xenc:EncryptedKey>
                <wsse:BinarySecurityToken ValueType="http://docs.
oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-
1.0#X509v3" EncodingType="http://docs.oasis-open.org/wss/2004/01/
oasis-200401-wss-soap-message-security-1.0#Base64Binary" wsu:Id="_
w4lpzuDj1hb1czr0ptHCeg22" xmlns:wsu="http://docs.oasis-open.org/
wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"\KE2ZA==</
wsse:BinarySecurityToken>
                <dsig:Signature xmlns="http://www.w3.org/2000/09/
xmldsig#" xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
                        <dsig:SignedInfo>
                                <dsig:CanonicalizationMethod
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                                <dsig:SignatureMethod Algorithm="http://
www.w3.org/2000/09/xmldsig#rsa-sha1"/>
                                <dsig:Reference URI="#BUHXr0pEKOjsguyWzr
KrHA22">
                                        <dsig:Transforms>
                                                <dsig:Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                                        </dsig:Transforms>
                                        <dsig:DigestMethod
```

```
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                                    <dsig:DigestValue>DLzrUyWQmjl+Hjh
z9ZEomPOLeaU=</dsig:DigestValue>
                              </dsig:Reference>
                              <dsig:Reference URI="#wdETzbUTVeZq50drbm
Wcsw22">
                                    <dsig:Transforms>
                                          <dsig:Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                                    </dsig:Transforms>
                                    <dsig:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                                    <dsig:DigestValue>OPseOpadhqa8VsB
HaH+ZLyPwG1g=</dsig:DigestValue>
                              </dsig:Reference>
                        </dsig:SignedInfo>
                        <dsig:SignatureValue>MWlJjuHw6y0L9YdEtclgYykuN
AtjEcScJK8l4y4ooAe+N4=</dsig:SignatureValue>
                        <dsig:KeyInfo>
                              <SecurityTokenReference wsu:Id="_
wfQx1sqYwS1KqOfQ0DPd9Q22" xmlns="http://docs.oasis-open.org/
wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd" xmlns:
wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd">
                                    <Reference URI="#_
w4lpzuDj1hb1czr0ptHCeg22"/>
                              </SecurityTokenReference>
                        </dsig:KeyInfo>
                  </dsig:Signature>
                  <wsu:Timestamp wsu:Id="wdETzbUTVeZq50drbmWcsw22"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-wssecurity-utility-1.0.xsd" xmlns="http://docs.oasis-open.org/
wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
                        <wsu:Created>2007-12-04T02:49:19Z</wsu:
Created>
                  </wsu:Timestamp>
            </wsse:Security>
      </soap:Header>
      <soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/" wsu:Id="BUHXr0pEKOjsguyWzrKrHA22" xmlns:wsu="http://docs.
oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd">
            <xenc:EncryptedData Type="http://www.w3.org/2001/04/
xmlenc#Content" Id="_4LDVnkQFr3DlbkZFa91f2w22" xmlns="http://www.
w3.org/2001/04/xmlenc#" xmlns:xenc="http://www.w3.org/2001/04/
xmlenc#">
```

```
                <xenc:EncryptionMethod Algorithm="http://www.
w3.org/2001/04/xmlenc#tripledes-cbc"/>
                <xenc:CipherData>
                      <xenc:CipherValue>\kzSqksyBm0e6LpxIDxYkJ+jccr
ldMuupLRYkqwIzasMTJ6rzVsfirYlQpaSUiJCq/SfyvOLiWLkPeEJlxmPig6o=</xenc:
CipherValue>
                </xenc:CipherData>
           </xenc:EncryptedData>
   </soap:Body>
</soap:Envelope>
```

Once the web services receive the above SOAP message, it will first decrypt it and then calculate the digital signature to find the match. Once the signature value matches, it is assumed that the data was not tampered with by anyone else. With Oracle WSM it is also easy to configure the decrypt and verify policy step to decrypt the message and then validate the signature. We will take a look at how to leverage Oracle WSM to both sign and encrypt and also to decrypt, and verify, in the next section with an example.

Sign and Encrypt by Example

In the web services world, it is not uncommon to exchange messages that are confidential, and at the same time, the integrity of the message should be protected. When both signature and encryption are involved, both the web service and the consumer application should be capable enough to sign, encrypt, decrypt and verify signatures. In this section, we will explore in detail how Oracle Web Services Manager can be leveraged to sign and encrypt, and also to decrypt and verify signature.

Example Overview

The example that we will be using in this chapter is the same time web service where the web service request is digitally signed and then encrypted. The web service will decrypt the data and then will validate the signature before processing the request. This can be summarized as:

- Time web service will validate if the incoming SOAP request is digitally signed and encrypted.
- Decrypt the message.
- Validate the signature of the decrypted data.

Consumer application (client application) will:

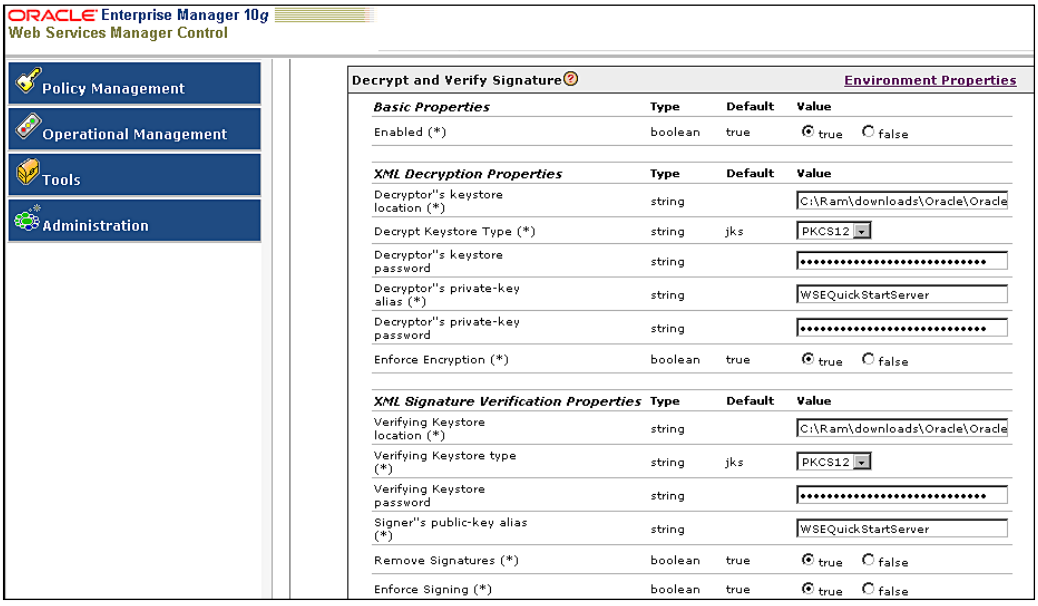
- Digitally sign the SOAP body of the request.
- Encrypt the SOAP body.

In this example, we will not be using any Microsoft application as a client application. We will use Oracle WSM to actually sign and encrypt the SOAP request to the time web service. By doing this, we will also be showing how Oracle WSM can be leveraged to centralize security operations of an external web service.

Time Web Service: Decrypt and Verify Signature

The web service, time service in our example, is configured to expect that the incoming SOAP request is digitally signed and then encrypted. The service should be able to decrypt and then validate the signature before the request is processed. Such configuration can be done easily with Oracle WSM by adding the Decrypt and Verify Signature policy step to the request message.

Once the web service is registered within Oracle Web Service Manager Gateway, the policy attached to the service can be modified to include the decrypt and verify signature. Let's consider for instance that the **Service ID** that was generated for this web service is `SID0030256`; the policy for this service can be modified to include the **Decrypt and Verify Signature** step (refer to the following screenshot).



In the figure, the decrypt and verify signature policy step is enabled and the SOAP request decrypted using the **PKCS12** certificate found at the specified location (Decryptor keystore location), with the keystore password of **test123**. The **Enforce Encryption** option describes that the encryption should be enforced.

The **XML Signature Verification Properties** section contains information about the location of the certificate, keystore password and the type of key store. This section also has the option to strictly enforce a signature and can also remove a signature value after validation.

Now that we have configured Oracle WSM to decrypt and verify the signature of an incoming SOAP message, we can now explore how Oracle WSM can be configured to sign and encrypt SOAP requests. The web service URL that is expecting a signed and encrypted SOAP request is `http://owsm.packtpub.com:3115/gateway/services/SID00256?WSDL`

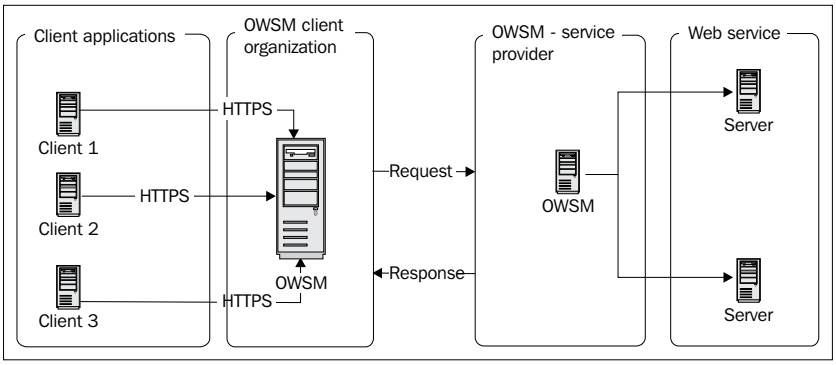
Beauty of Oracle WSM Gateway: Sign And Encrypt by Oracle WSM

A client application such as Microsoft .NET or Java should digitally sign the SOAP message and encrypt before invoking the web service. However, in order to sign and encrypt a SOAP message, the client application should:

- Manage the private key of the client organization/application.
- Manage the public key of the service provider organization.

When an organization is digitally signing all SOAP messages, and when there are multiple client applications, it becomes a challenge and a security risk to manage the private key deployment across all the applications.

With Oracle Web Services Manager, service consumer organization can centrally manage all the security policies of the web services that it interacts with. This provides a scalable architecture where all the client applications can interact with Oracle WSM Gateway and the gateway can enforce the security policy. The following diagram show the architecture where Oracle WSM is used in both service provider and service consumer ends.



In the diagram, the client applications at the service consumer end talk to Oracle WSM Gateway over HTTPS and the Oracle WSM talks to the web service at the service provider end. The Oracle WSM at the service provider will enforce the security policies before the service is executed.

Note: Instead of the gateways, you can either use Oracle WSM client agents or any other third party software that supports WS-security standards (e.g. WSE from Microsoft, Glassfish from Sun, etc.)

In our example, the time web service is protected by Oracle WSM which will enforce the security policy that the incoming SOAP request should be digitally signed and encrypted. The following steps describe how Oracle WSM can be leveraged at both service provider and consumer side to enforce security policy to access the time service.

Service Provider:

- Web service is deployed inside the application server.
- Web service is registered with Oracle Web Services Manager Gateway.
- Policy is modified to decrypt and validate signature.
- The new URL (Service ID of Oracle WSM) is published to the consumer. (In our example it will be `http://owsm.packtpub.com:3115/gateway/services/SID00256?WSDL`)

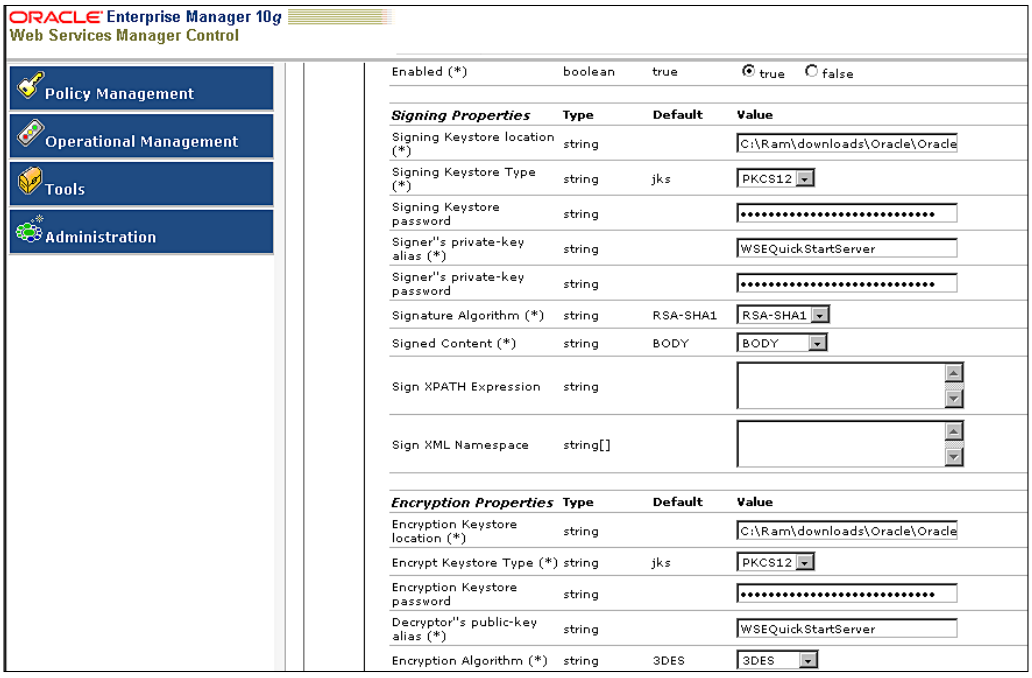
Service Consumer:

- Service consumer is also using Oracle Web Services Manager.
- The WSDL URL is registered within Oracle WSM.
- Oracle WSM will create another URL with the service ID.
- Policy is modified to sign the message and then encrypt:
    - The Request policy step is modified to include the sign and encrypt policy step.
- Client application such as .NET or Java will invoke the new web service URL (for example, it could be SID00259?WSDL).

Note: In this example, this can be tested within the same Oracle WSM installation.

Sign And Encrypt Policy

The web service URL is registered within Oracle WSM and then the policy is modified to sign and encrypt the message. The web service URL, `http://owsm.packtpub.com:3115/gateway/services/SID00256?WSDL` is now registered within Oracle WSM. The Request policy step of the web service policy in Oracle WSM is modified to include the **Sign and Encrypt** policy step. The following screenshot shows the steps to include the **Sign and Encrypt** policy step.



In the figure, the **Signing Properties** section has the key store location, the key store type, the password of the key store and the private key to digitally sign the message entries. The **Encryption Properties** section contains the information regarding the public key of the service provider, key store, and the password to encrypt the message.

Once the policy is created, when a client application makes the request, Oracle WSM will sign and encrypt the SOAP request message. The client application invokes the web service (in our example you can actually use Test Page from the Tools section by entering the WSDL URL and then just clicking invoke) invokes the web service:

- Client application invokes the new WSDL URL that points to the Oracle WSM Gateway.
- Oracle WSM Gateway will invoke the Request policy step and will sign and encrypt the message.
- Since the time web service URL is registered within Oracle WSM Gateway, the client gateway will invoke the time web service, which is signed and encrypted.
- Oracle WSM at the service provider will decrypt and validate the signature
- The request is passed to the actual time web service.
- Time web service will respond with the actual time.

Summary

In this chapter, we described two different things: one is how to digitally sign and encrypt the message (and decrypt and verify the signature); and also how Oracle WSM can be used at the consumer side to enforce or attach the required security credentials before invoking the web service.

13

Enterprise Security — Web Services and SSO

Web services security addresses the concerns regarding authentication, authorization, confidentiality and integrity, and Oracle Web Services Manager makes it easier to implement security in web services. However, in any organization, web services security is not the only security solution; it's probably one of the many security solutions they might have. Many times you have multiple LDAP directories, multiple authentication data stores, web access management (SSO) solutions, etc. When deploying the web services security solution, one should consider the enterprise security to provide an integrated security solution. In this chapter, we will take a closer look at the integrated web services security solution.

Web Services Security Components

By now you must be familiar with various components of web services security, such as authentication, authorization, encryption and signature. Before web services were widely adopted in organizations, there were different flavors of security systems that could authenticate and authorize users. Most of the authentication and authorization systems leverage LDAP compliant directory stores, such as Active Directory, Sun One, Novell eDirectory, etc. However there are systems that authenticate users against a traditional database such as Oracle or SQL Server. Authorization is either implemented as role-based access control, where role/group information is stored in the LDAP directory, or a custom implementation where user's privileges are checked against certain attributes or permissions.

Now with the advent of web services, the same user or system can be authenticated against existing credential store (be it database or active directory or eDirectory, etc.) before they can gain access to the web services. In order to reduce the support cost and leverage investment in an existing credential store, web services security components, i.e. authentication and authorization should be integrated with existing enterprise security systems. In the next few sections we will discuss:

- Leveraging investment in existing credential stores i.e. directories and databases.
- Integrating with access management solutions.
- Security token service.
- Integrated web services security architecture.

Authentication, Authorization and Credential Stores

Before web services were even introduced in the organization, there were various systems that authenticate and authorize users against a credential store, typically an LDAP compliant directory or database. Applications that take advantage of existing credential store, to authenticate its users now have to face the challenge of authenticating users who need access to its web services.

Oracle Web Services Manager supports various authentication options, including authenticating users against a file, active directory, or any other LDAP compliant directory. Maintaining a separate directory or file system to authenticate users to gain access to web services will become an additional maintenance effort.

In order to reduce the cost of maintaining a list of separate credential stores for each web service that is registered within Oracle WSM, it can actually authenticate users against existing credential stores, such as active directory or any other LDAP compliant directory. If an application is authenticating its users against a database and would like to authenticate the web services as well against the database, a custom authentication mechanism can be written to authenticate against the database.

Note: When organizations have different LDAP directories and databases as their credential stores, they can all be combined to form a virtual directory tree using any virtual directory product such as Oracle Virtual Directory, Radiant Logic, Symlabs, etc. Once a virtual directory layer is created, Oracle WSM can authenticate against that directory (all virtual directory products expose LDAP v3 compliant interface). Virtual directory topic is outside the scope of this book.

Integrating with Web Access Management Solution

Web access management solutions are deployed primarily to:

- Provide a single sign-on experience to end users.
- Centrally manage resources and their access policies to enforce better access control.

While the access management product eliminates multiple username and passwords and centralizes access control enforcement, they are mainly intended for web applications. On the other hand, web services were being deployed to integrate better with existing applications. The security of the web services was often left to be implemented either with the web service provider or with a web service security product such as Oracle Web Services Manager. What makes it more interesting is that the web services were being invoked from the web applications that are protected by web access management products.

Let's take a closer look at how the web access management products enforce access control for web applications. A Typical web access management product has the following components:

- Policy server to manage the access policies.
- Policy enforcement point where the access policies are enforced.

A typical policy enforcement point is usually an ISAPI filter or agent sitting on the web server which will:

- Intercept the request to web application.
- Validate if the user is authorized to access or not.
- Validate if the user is authenticated, if it requires authorization.
- Present the user with a login page, if he is not authenticated.
- Validate the credentials the user has entered against the data store
- Create an encrypted cookie (i.e. single sign on cookie).
- Validate whether the user is authorized to access the requested resource (i.e.: web page).
- Allow the user to perform the action (e.g: view the page) if user has privileges.

A web services security product such as Oracle WSM also performs similar steps to make sure that the user has enough privileges before the web service operation is performed. Oracle WSM will:

- Extract credentials from the web service message (typically from SOAP header).
- Validate the credentials.
- Validate if the user has appropriate privileges to perform the operation.
- Allow the web service operation, if yes.

Now let's take a closer look at where a web service is invoked from a web application. In this case, the web application is protected (i.e. security is enforced) by a web access management product. Let's consider an example where a credit card payment web service is invoked from a shopping web site (or a customer update web service is invoked from a CRM application). In order to effectively secure both the web application and the web service, both should authenticate, i.e. validate the end user before any operation is allowed. The web application is protected by the web access management product, which will validate the user credential and then create a SSO token (i.e. cookie). The challenge is that when the web service is invoked on a button click from the web application, the web service has to authenticate (or identify) the end user and not the server/web application that is making the web service call. It can be easily addressed by using the SSO token to authenticate the web service access.

Authentication based on the SSO token (i.e. SSO cookie) actually leverages the existing investment in web access management platforms and also provides a means to centralize the access policies for web services as well. When SSO token is used to authenticate the web services, the various interactions that happen between the user, the web application, web access management, and Oracle WSM are:

- Web application protected by Web Access Management (WAM) product such as Oracle Access Manager (or Siteminder).
- User enters credentials (username and password).
- WAM product will validate the credentials.
- WAM product will create the SSO token (i.e. SSO cookie).
- Web service is invoked by the web application (on an event such as button click, form post, etc.)
- Oracle WSM client agent (or custom code) will insert the SSO token in the SOAP message.
- Oracle WSM will validate the SSO token using WAM SDK (this will ensure that the SSO token is valid and that the user has permission to access the web service).

Let's take a closer look at the architecture that leverages both Oracle WSM and Oracle Access Manager (or Siteminder) in the same organization to protect the web applications and web services. In this scenario, the various steps involved to enforce the access control will be:

- Install and configure web access management product.
- Configure the resource policies for the web application.
- Install and configure Oracle Web Services Manager.
- Configure the gateway, policy, etc.
- Install the Oracle WSM client agent on the client application (or a custom written one):
    - Client agent would insert the SSO token in the SOAP message.
- Install web access management SDK where the Oracle WSM is installed:
    - SDK and custom code would let you validate the SSO token against the web access management platform.

The various advantages of using SSO token to authenticate the web services are:

- Web service provider can authenticate the actual end user, not the web application that is acting as service consumer.
- Access policies can be configured within the web access management product itself (thus removing any additional access policy maintenance within Oracle WSM).

However, the disadvantages to using SSO token to authenticate web services are:

- Authentication is not based on any interoperable standards.
- Client applications that cannot provide the SSO token will be unable to consume the web service.
- Solution is only limited to the same internet domain and requires Oracle WSM and Oracle Access Manager.

The disadvantages actually introduce the challenge in integrating the web access management system with the web services security product, such as Oracle WSM, for authentication and authorization. In the next section we will look at how to overcome the challenge by means of Security Token Service.

Security Token Service: Bridging the GAP between WAM and Oracle WSM

One of the key challenges in integrating Oracle WSM with Oracle Access Manager (or any other web access management product) is that the each client (i.e. web service consumer) should have Oracle WSM client agent to insert the SSO token, and it does not provide any flexibility for web services to be consumed by someone who does not support Oracle WSM client agents.

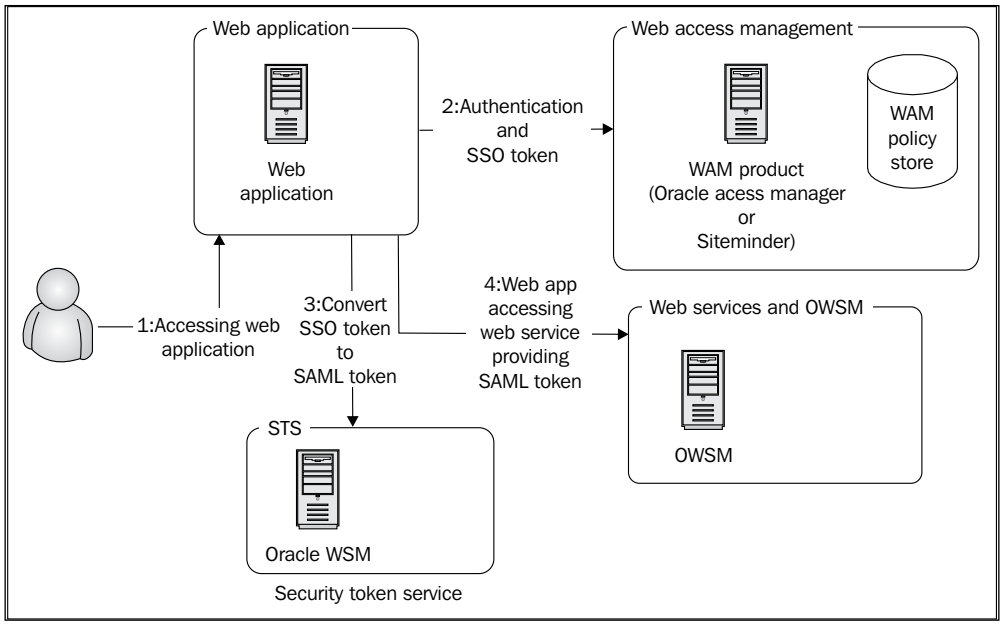
The tight integration between Oracle WSM and web access management reduces the flexibility of expanding the service adoption with applications within organizations that cannot provide SSO cookie (Oracle Access Manager). One way to overcome this challenge is to break the tight integration and make it loosely coupled, but still support industry standards such as **Security Assertion Markup Language (SAML)**.

SAML is the interoperable industry standard that is widely used to federate identities across organizational boundaries. Web services security specification also adopted SAML as one of the authentication token profiles, and it is also supported by Oracle Web Services Manager.

In Oracle Web Services Manager, you can configure the authentication step to validate the SAML token, or you can write custom steps to validate any other token profiles that are part of WS-security. Once Oracle WSM authenticates, it can then authorize the users against Active Directory or any other directory store.

Now we know that web access management creates SSO token to track the authenticated users and Oracle WSM can authenticate based on SAML token profile (or other token profiles such as **UserName** token), we can integrate leverage Oracle WSM to convert the SSO token to SAML token and let the web services authenticate using SAML token.

Security token service gives flexibility to convert one token format to another format, so that it can bridge the gap between web access management systems and web services security products, such as Oracle WSM. The following figure show the interaction between user, web access management, STS and Oracle WSM.



The figure show that Oracle WSM is used a security token service to convert the Web Access Management SSO token to a SAML token. SAML token is used by another Oracle WSM gateway or agent or native application to authenticate access to the web service. The advantages of using security token service are:

- Centralizes the token translation in one place (SAML 1.1 to SAML 2.0, SSO token to SAML, etc.)
- Gives flexibility for Oracle WSM to accept SAML and other types of token format without requiring all the clients to send the SSO token in a custom SOAP extension.
- End user identity is flown from web access management to Oracle WSM (e.g. John Doe login with "JDoe" and password; web access management creates a SSO token; Oracle WSM understands John Doe by means of SAML with JDoe as identifier, or UserName token with JDoe as identifier).
- Web applications can be protected by WAM products such as Siteminder, Sun Java Access Manager, Oracle Access Manager, etc.

You can use Oracle WSM just as a token service to convert tokens from SSO tokens (OAM or Siteminder) to SAML or any other tokens.

Integrated Security Architecture

Enterprise security architecture design should consider passing the end user identifier across all the layers, such as web application to back-end components, to web services, etc. The architecture design should also keep in mind the interoperability, and having a service to broker the authentication types should be an integral part of enterprise security architecture. The integrated security architecture design should consider having security token service to bridge the gap between web access management product and Oracle WSM. In this section, we will explore in detail how to use Oracle WSM to issue SAML tokens.

While security token service is based on WS-Trust specification, there are times where there is a need to convert from one token format to another token format when Oracle Web Services Manager is the only WS-Security product in the organization. In this section, we will take a closer look at Oracle WSM to issue SAML tokens.

Consider the scenario where your organization has Oracle Web Services Manager and your web application should access a web service outside of your organization (e.g. Amazon web service) that requires a SAML token profile to authenticate the user. The user interactions can be described as:

- User will authenticate with username and password to the web application.
- Web application has to invoke external web service.
- Web service requires a SAML token with user information.

While authenticating the user to the web application can be handled by custom or web access management products, Oracle WSM can be used to invoke the web service by attaching the SAML token profile. In order for Oracle WSM to issue a SAML token, we have to work around the way web services are imported into Oracle WSM. In an ideal situation, the external web service will be registered within Oracle WSM Gateway or agent, and the client application or client agent would be required to attach the SAML token.

Here we would like to demonstrate that Oracle WSM can be used to issue a SAML token without using client agents. In this case, we can do the following within Oracle WSM to issue the SAML token:

- Register the external web service within Oracle WSM gateway.
- Oracle WSM gateway will now create a new URL with a new WSDL description.

- Policy can be configured as:
    - The first step of Request pipeline can extract credential and validate username and password.
    - The last step of Request pipeline can be configured to insert SAML token profile.

Now the client application will invoke the internal Oracle WSM WSDL URL with their internal username and password. Oracle WSM will validate the username and password, and then attach a SAML token while accessing the external web service.

Integrated security architecture that bridges the gap between WAM and Oracle WSM will increase the security of the enterprise applications by passing the end user information across the application boundaries.

Summary

Web services security products such as Oracle WSM are focused on externalizing the security operations from the web service providers and consumers. Enterprise architecture should leverage products such as Oracle WSM to externalize the security operations such as authentication, authorization, encryption and signature, and at the same time support industry standards such as SAML for authentication. This chapter focused on integrating Oracle WSM with existing web access management products with or without using security token service. This book focused on how to use Oracle Web Services Manager to authenticate and authorize access to web services and also to protect the confidentiality and integrity of messages.

Index

P

S

T

W

X

Thank you for buying Oracle Web Services Manager

About Packt Publishing

Packt, pronounced 'packed', published its first book "*Mastering phpMyAdmin for Effective MySQL Management*" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: `www.packtpub.com`.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to `authors@packtpub.com`. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.