## Lab 8: How to implement and monitor SOAP request and response packets. Analyze parts of it and compare them with the operations (java functions) headers

What is SOAP?

SOAP, which stands for Simple Object Access Protocol, is a protocol for exchanging structured information in web services. It is a protocol specification that defines a set of rules for structuring messages that can be exchanged between computers over a network. SOAP is used as a foundation for a variety of web services, especially in enterprise-level applications and distributed systems.

Key characteristics of SOAP include:

Message Format: SOAP messages are XML-based, meaning that they are written using XML syntax. The structure of a SOAP message typically includes a header and a body. The header can contain information about the message, such as authentication details or other metadata, while the body contains the actual data being transmitted.

Protocol Independence: SOAP messages can be sent over a variety of protocols, including HTTP, SMTP, or even more specialized protocols. The most common transport protocol for SOAP is HTTP.

Extensibility: SOAP is extensible and allows for the use of additional specifications and standards to enhance its functionality. For example, WS-Security is a standard that adds security features to SOAP messages.

Platform Independence: SOAP is designed to be platform-independent, meaning that it can be used with applications developed on different platforms and using different programming languages.

Language Neutrality: SOAP allows for communication between applications written in different programming languages. It relies on a standardized XML format for data exchange, making it language-neutral.

Stateless: SOAP itself is stateless, meaning that each request from a client to a server is treated as an independent message.

Web Services: SOAP is often used as the messaging protocol for web services. Web services allow applications to communicate and share data over the internet, and SOAP is one of the protocols that facilitate this communication. To monitor SOAP request and response packets in a Java program, you can use a combination of Java APIs and libraries. One way to achieve this is by using a tool like Apache CXF, which is a popular open-source web services framework that provides support for SOAP-based services.

Below is a simple Java program that uses Apache CXF to intercept and analyze SOAP request and response packets. The program includes a custom SoapInterceptor that prints the SOAP message

details, and it compares the SOAP operations (functions) headers. Please note that this is a basic example, and you may need to adapt it based on your specific requirements.

<soap:Envelope xmlns:soap="<http://www.w3.org/2003/05/soap-envelope>" xmlns:example="[http://www.example.org">](http://www.example.org/)

<soap:Header>

<!-- Header information (optional) -->

</soap:Header>

<soap:Body>

<!-- Payload/data of the message -->

<example:HelloWorldRequest>

<example:Name>John</example:Name>

</example:HelloWorldRequest>

</soap:Body>

</soap:Envelope>

In this example, a SOAP message is being sent with a "HelloWorldRequest" payload, which includes a "Name" element with the value "John." The actual structure of the message would depend on the specific requirements of the web service being used.

To monitor SOAP request and response packets in a Java program, you can use a combination of Java APIs and libraries. One way to achieve this is by using a tool like Apache CXF, which is a popular open-source web services framework that provides support for SOAP-based services.

Below is a simple Java program that uses Apache CXF to intercept and analyze SOAP request and response packets. The program includes a custom SoapInterceptor that prints the SOAP message details, and it compares the SOAP operations (functions) headers. Please note that this is a basic example, and you may need to adapt it based on your specific requirements.

import org.apache.cxf.binding.soap.SoapMessage;

import org.apache.cxf.binding.soap.interceptor.AbstractSoapInterceptor;

import org.apache.cxf.binding.soap.interceptor.SoapInterceptor; import org.apache.cxf.endpoint.Client;

import org.apache.cxf.endpoint.Endpoint; import org.apache.cxf.frontend.ClientProxy; import org.apache.cxf.interceptor.Fault;

import org.apache.cxf.jaxws.endpoint.dynamic.JaxWsDynamicClientFactory;

import javax.xml.namespace.QName;

public class SoapMonitor {

public static void main(String[] args) { try {

// Define the SOAP service endpoint URL

String serviceUrl = "[http://example.com/YourSoapService?wsdl"](http://example.com/YourSoapService?wsdl);

// Create a dynamic client using Apache CXF

JaxWsDynamicClientFactory dcf = JaxWsDynamicClientFactory.newInstance(); Client client = dcf.createClient(serviceUrl);

// Attach a custom SoapInterceptor to the client attachInterceptor(client);

// Perform a sample SOAP operation invokeSoapOperation(client, "yourOperationName");

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

}

}

private static void attachInterceptor(Client client) {

// Attach a custom SoapInterceptor to the client Endpoint endpoint = client.getEndpoint();

endpoint.getInInterceptors().add(new CustomSoapInterceptor());

}

private static void invokeSoapOperation(Client client, String operationName) { try {

// Create a QName for the operation

QName operationQName = new QName("<http://example.com/namespace>", operationName);

// Invoke the SOAP operation

Object[] result = client.invoke(operationQName);

// Process the result if needed

// ...

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

}

}

// Custom SoapInterceptor to monitor SOAP messages

public static class CustomSoapInterceptor extends AbstractSoapInterceptor implements SoapInterceptor {

public CustomSoapInterceptor() { super("phase");

}

@Override

public void handleMessage(SoapMessage message) throws Fault {

// Access and analyze the SOAP request and response String requestContent = message.getContent(String.class); System.out.println("SOAP Request:\n" + requestContent);

String responseContent = (String) message.getExchange().getOutMessage().getContent(String.class);

System.out.println("SOAP Response:\n" + responseContent);

// Compare with the operations (functions) headers compareWithOperationsHeaders(message);

}

private void compareWithOperationsHeaders(SoapMessage message) {

// Implement your logic to compare SOAP headers with operations (functions) headers

// ...

}

}

}

In this example:

The JaxWsDynamicClientFactory is used to create a dynamic client for the SOAP service.

A custom SoapInterceptor (CustomSoapInterceptor) is attached to the client's endpoint to intercept SOAP messages.

The handleMessage method in the interceptor is invoked when a SOAP message is sent or received. You can analyze the request and response messages in this method.

The compareWithOperationsHeaders method is a placeholder for your logic to compare SOAP headers with operations (functions) headers.

Make sure to replace the placeholder values like the service URL, namespace, and operation name with your actual service details. Also, adjust the logic inside compareWithOperationsHeaders to suit your specific requirements for comparing headers.