1. **What is restful web services ?**

**REST** stands for Representation State Transfer . Restful webservices are Rest architecture based web services . In Rest architecture every content is treated as resource . These resources can be accessed via URI. Unlike SOAP webservice , it doent have its own security standards instead it derive its from underlying protocol. It use the HTTP protocol for data communication. Refer the web application security standards to make the restful services secure such as , Input validation, session based authentication , no sensitive data in the uri like user name , password and session tocken , Restriction on method execution, validate malformed XML and JSON , throw generic error messages.

Restwebservices are lightweight (no extra xml markup required) and highly scalable and easy maintanable and easy to build(no toolkits required) . These are used to create API’s for web based application. There are various representations of resource supported as XML,plain text and JSON. The most popular representation used are XML and JSON. The resource is REST similar to Object in Object oriented programming or Entitty in the database.

Rest Server simply give access to the resources , rest client can access and modify the resources.

The following Http methods commonly used in the Rest web Services.

**Get –** It is provide the read only access to the resource

**Put-**  it is used to create the new resource

**POST-** it is used to update an exisiting resource or create the new resource

**Delete –** it is used to remove the resource

**Options –** it is used to get the supported operations on the resource

**Ex:**

package com.tutorialspoint;

import java.util.List;

import javax.ws.rs.GET;

import javax.ws.rs.Path;

import javax.ws.rs.Produces;

import javax.ws.rs.core.MediaType;

@Path("/UserService")

public class UserService {

UserDao userDao = new UserDao();

@GET

@Path("/users")

@Produces(MediaType.APPLICATION\_XML)

public List<User> getUsers(){

return userDao.getAllUsers();

}

}

@path – is used to specify the path of the web service and resource

@Path(“/UserService”) – is the path of the web service to be accessed

@Path(“/users”) – is the path of the resource to be accessed

@GET – is the method to be used to access the resource

@Produces - Specifies the MIME media type(data format) of the response that the resource can produce and send back to the client .

@Produces(MediaType.APPLICATION\_XML,MediaType.APPLICATION\_JSON) – it produces XML response also JSON response. @Produces applied to both class level and method level. If it is applied to class level all the methods return the same MIME media type specified at class level. If it is applied at method level , it overrides the class level settings.

The above web service can be accessed via

http://localhost:8080/UserManagement/rest/UserService/user

protocol://hostname:portnumber/servicename /urlpattern/resourcetype / resourceid

**Idempotent :-** Results will be same always howmany times we hit the same request.

In the above Ex, getUsers() method is accessing the userDao.getAllUsers()

Lets define the UserDao here,

package com.tutorialspoint;

import java.util.ArrayList;

import java.util.List;

public class UserDao {

public List<User> getAllUsers(){

List<User> userList = null;

//DB login to fetch the uswer from database , for temporarily just construct the User here

User user = new User(1, "Mahesh", "Teacher");

userList = new ArrayList<User>();

userList.add(user);

return userList;

}

}

From the above code , the getAllUsers() method constructing the list of User and returning it.

Lets construct the User object or Entity,

package com.tutorialspoint;

import java.io.Serializable;

import javax.xml.bind.annotation.XmlElement;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name = "user")

public class User implements Serializable {

private static final long serialVersionUID = 1L;

private int id;

private String name;

private String profession;

public User(){}

public User(int id, String name, String profession){

this.id = id;

this.name = name;

this.profession = profession;

}

public int getId() {

return id;

}

@XmlElement

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

@XmlElement

public void setName(String name) {

this.name = name;

}

public String getProfession() {

return profession;

}

@XmlElement

public void setProfession(String profession) {

this.profession = profession;

}

}

The above entity itself annotated with @XmlRootElement and @XmlElement. These are JAXB annotations as described below

@XmlRootElement(name=”name” or namespace=”namespace”) - specifies the Root element of the XML. It is the class level annotation

@XmlType(proporder={field1,field2}) – specifies the order of the element in the XML. It is the class level annotation

@XmlElement (name=”eleemntname” )– specifies the element of the XML. it is the method level annotation.

Ex:

package de.vogella.xml.jaxb.model;

import javax.xml.bind.annotation.XmlElement;

import javax.xml.bind.annotation.XmlRootElement;

import javax.xml.bind.annotation.XmlType;

**@XmlRootElement(name = "book")**

// If you want you can define the order in which the fields are written

// Optional

**@XmlType(propOrder = { "author", "name", "publisher", "isbn" })**

public class Book {

private String name;

private String author;

private String publisher;

private String isbn;

// If you like the variable name, e.g. "name", you can easily change this

// name for your XML-Output:

**@XmlElement(name = "title")**

public String getName() {

return name;

}

// other getters and setters

}

As per REST architecture, restful web services are stateless.

**JAX-RS** is a java based API useful for creating the Rest ful web services and clients. This API supports various annotations as below for restful web service creation.

1. **@Produces -** specifies the response type of the resource (web service method)
2. **@Consumes –**specifies the request type to be consumed by the resource
3. **@QueryParam –** it is used to inject the url query string parameter with the method parameter

Ex: “**users/query?from=100&to=200&orderBy=age&orderBy=name**”

import java.util.List;

import javax.ws.rs.GET;

import javax.ws.rs.Path;

import javax.ws.rs.QueryParam;

import javax.ws.rs.core.Response;

@Path("/users")

public class UserService {

@GET

@Path("/query")

public Response getUsers(

**@QueryParam("from")** int from,

**@QueryParam("to")** int to,

**@QueryParam("orderBy")** List<String> orderBy) {

return Response

.status(200)

.entity("getUsers is called, from : " + from + ", to : " + to

+ ", orderBy" + orderBy.toString()).build();

}

}

1. **@PathParam –**  it is used to inject the parameter value from the uri, that is defined in the @path expression into the java method arguments.

Ex:

import javax.ws.rs.GET;

import javax.ws.rs.Path;

import javax.ws.rs.PathParam;

import javax.ws.rs.core.Response;

@Path("/users")

public class UserRestService {

@GET

@Path("**{year}/{month}/{day}")**

public Response getUserHistory(

**@PathParam("year")** int year,

**@PathParam("month")** int month,

**@PathParam("day")** int day) {

String date = year + "/" + month + "/" + day;

return Response.status(200)

.entity("getUserHistory is called, year/month/day : " + date)

.build();

}

}

1. **@FormParam –** It is used to inject the HTML form parameters to the java method arguments.

**Ex:**

import javax.ws.rs.FormParam;

import javax.ws.rs.POST;

import javax.ws.rs.Path;

import javax.ws.rs.core.Response;

@Path("/user")

public class UserService {

@POST

@Path("/add")

public Response addUser(

**@FormParam("name")** String name,

**@FormParam("age")** int age) {

return Response.status(200)

.entity("addUser is called, name : " + name + ", age : " + age)

.build();

}

}

1. **@CookieParam :-** it is used inject the entire cookie or cookie parameter value with the method.

**Ex: 1**

@GET

**public** Response getCookie(@CookieParam(**"name"**) Cookie cookie){

**return** Response.ok().build();

}

@GET

@Path(**"/value"**)

**public** Response getCookieValue(@CookieParam(**"name"**) String cookie){

**return** Response.ok().build();

}

1. **@HeaderParam :-** It is used to inject the specific header value to the method parameter. It can also been done by @Context ,but @Context HttpHeaders inject the entrie header into method.(refer @context down)

**Ex:**

@Path("/users")

public class UserService {

@GET

@Path("/get")

public Response addUser**(@HeaderParam("user-agent")** String userAgent) {

return Response.status(200)

.entity("addUser is called, userAgent : " + userAgent)

.build();

}

}

It prints the user-agent header value.

1. **@MatrixParam:-** It is used to inject the parameter value comes along with path but separated by semicolon. It is an key-value pairs

**EX:** “**/books/2011;country=malaysia;author=mkyong**”

@Path("/books")

public class BookService {

@GET

@Path("{year}")

public Response getBooks(**@PathParam("year")** String year,

@**MatrixParam("author")** String author,

@**MatrixParam("country")** String country) {

return Response

.status(200)

.entity("getBooks is called, year : " + year

+ ", author : " + author + ", country : " + country)

.build();

}

}

Year =2011

Author=mkyoung

Country=malaysia

1. **@Context :-** it is an alternative method of @QueryParam used to inject the paramter value from URI to method local variable. It is also called as programmetic way of receiving the uri query parameter. Basically it retrieves the context of the resource ie. Request context. The entire URI is injected to method. It is also used to inject the http header into method.

**Ex: 1**

@Path("/users")

public class UserService {

@GET

@Path("/query")

public Response getUsers**(@Context UriInfo** info) {

String from = info.getQueryParameters().getFirst("from");

String to = info.getQueryParameters().getFirst("to");

List<String> orderBy = info.getQueryParameters().get("orderBy");

return Response

.status(200)

.entity("getUsers is called, from : " + from + ", to : " + to

+ ", orderBy" + orderBy.toString()).build();

}

}

**Ex: 2**

@GET

@Path("/get")

public Response addUser(@**Context HttpHeaders** headers) {

String userAgent = headers.getRequestHeader("user-agent").get(0);

return Response.status(200)

.entity("addUser is called, userAgent : " + userAgent)

.build();

}

1. **@DefaultValue :-** It is used to specify an default value for the optional parameter.

**Ex:**

@Path("/users")

public class UserService {

@GET

@Path("/query")

public Response getUsers(

**@DefaultValue("1000")** @QueryParam("from") int from,

**@DefaultValue("999")@**QueryParam("to") int to,

**@DefaultValue("name")** @QueryParam("orderBy") List<String> orderBy) {

return Response

.status(200)

.entity("getUsers is called, from : " + from + ", to : " + to

+ ", orderBy" + orderBy.toString()).build();

}

}

1. **@Path :-** specifies the path to the resource and web service
2. **@Get :-**

**Ex:** <http://localhost:8080/UserManagement/rest/UserService/users> - retrieves list of users

<http://localhost:8080/UserManagement/rest/UserService/users/1> - retrieve the user of id 1

1. **@ Post :-**

**Ex:** <http://localhost:8080/UserManagement/rest/UserService/users/2> - Update the user with id 2

1. **@Delete:-**

**Ex:** <http://localhost:8080/UserManagement/rest/UserService/users/1> - delete the user with id 1

1. **Put:-**

**Ex:** <http://localhost:8080/UserManagement/rest/UserService/users/2> - Insert he user with id 2

1. **Options:-**

**Ex:** <http://localhost:8080/UserManagement/rest/UserService/users> - list out the supported operations in web service

1. **@Head:-**

**Ex:** <http://localhost:8080/UserManagement/rest/UserService/users> - retrieves http header

**Advantages :**

1. **Caching :**

Rest provides good caching mechanism.

**What is caching ?**

Caching means storing the server response in the client side. So that client no need to make the subsequent request for the same resource again and again. The server response itself should have information how to cache the response. So that this would be cache for the time period or not cached .

The following the headers available in HTTP server response with respect to caching.

**Date :-** The date when the resource is created.

**Last Modified:-** The date and time when the resource is modified at last

**Cache-Control:-** Primary header to control caching

**Expires:-** Expiration date and time of caching

**Age:-** Duration in second from when the resource is fetched from server.

**Cache-Control Headers:**

**No-cache/ no store : -** indicates the resource is not cashable

**Private :- I**ndicates that only that particular client and server can cache the resource not the intermediary

**Public :-** indicates that any one can catch the resource

**Max-age:-** indicates that the validity of the cached resource in seconds. After this the client can make another request

**Must-revalidate:**- indicates to server to revalidate the resource if max-age has passed.

**Ex:**

Tutorials point web site caching the pages and when we hit the send time (without network connectivity) still it shows the cached pages.

1. **Stateless:**

Restful web services provide stateless invocation. So the server not required to store anything across any request. So There will not much communication between server makes it horizantally scalable (refer horizantal scaling below). So that load balancer easily route the request to different servers.(Get request into one server and Post request into other server without replicating the session states among these servers). But SOAP relies on state and session , so we have to use the smarter load balancer to redirect the client request for process or should replicate the sessions state among the server . So SOAP is for stateful invocation.

1. **High Scalabiltiy:**

Because of the caching and statelessness behaviour of the Restful web services make this highly scalable in horizantally.

1. **Lightweight:**

Restful webservices support the plain text , xml and JSON message formats. It doent wrap the messages with some other standard . But SOAP web services , the xml messages are wrapped into SOAP Envelope (SOAP standard) which adds another layer on the message. It makes SOAP as heavyweight.

1. **Support of multiple data formats:**

Restful web services supports multiple data formats such as XML, plain text and JSON .

1. **Fast Development:**

It is easy to develop , no toolkit required for development and learning curve is very short. But SOAP is difficult to implement , it requires toolkit for development. Good learning is required about SOAP standards.

1. **Good Performance:**

Since its light weight and the result are human readable form , its giving good performance.

But in SOAP the result are not human readable form , it needs serialization of converting the xml to object and object to xml.

**Disadvantages:**

Restful web services has some disadvantages over SOAP.

1. **Security:**

It doesn’t have its own standards for security. It always rely on http standars on security. But SOAP has Security standards as WS-Security

1. **Protocol Support:**

It supports only HTTP protocol. Tied to HTTP transport protocol .But SOAP is designed to support for multiple protocols such as HTTP ,TCP , UDP and SMTP.

1. **Error Handling:**

It doesn’t have its own error handling standards. It depends on HTTP error handling standars . But SOAP have its own error handling standard as SOAP Fault.

1. **When to use REST web service over SOAP ?**

Choose SOAP :

1)When we have formal contracts

2) When stateful conversations are required

3) Asynhronous Processing and invocation

4) Learning curve takes time

5) it needs toolkit for development efficiency(WSDL dev)

6) When application needs end-to-end security or absolute distributed transational reliability

7) when only XML can be supported

Choose Rest:

1. When Limited bandwidth and resources
2. When need statelesss processing
3. Caching situations
4. Learning doesn’t take much time
5. It doesn’t require the tool kit for development
6. It supports only point-to-point communication.
7. Multiple data formats to be supported
8. Rest supports high scalability than SOAP(Restricted get is supported but not in SOAP) for web application with high load.
9. Rest messages are smallar in size and consume lesser bandwidth
10. Rest provides better caching so it increase performance
11. Errorhandling

Both are reliable and flexible approach to expose the functionality in distributed environment. Both has advantages and disadvantages .But still we need to conside the below two points whne selecting the best approach.

1. Type of clients supported. Rest provides an effective way for interacting with lightweight clients as mobile phones
2. Flexibility and standardization acccepted by the organizations culture.
3. Architecture requirement (end-to-end or point-to-point)

**3) What are micro services?**

Micro services or micro services architecture is an architecture style which has collection of loosely coupled services which implement business functionalities. It enable the continuous development of delivery/deployment of large , complex applications. It enables the organization to evolve its technology stack.

**Why microservices ?**

To overcome the issues and limitations in monolithic architecture pattern. Monolithitic architecture pattern is an traditional architectural style which is still used in many of the large , complex applications development. But it has some limitations and issues ,so we are moving to microservices architecture pattern.

**What is Monolithitic architecture pattern?**

Refer the link : [**http://microservices.io/patterns/monolithic.html**](http://microservices.io/patterns/monolithic.html)

The components in an application are tightly coupled and deployed as single war file is known as monolithic application.

**When to choose monolithic architecture?**

**Scnerio:**

When we need to develop an server-side enterprise application , it should have different types of clients such as desktop browsers, mobile browsers , native mobile applications. It should expose an API to its third party consumers. It should interact with other system through web services or message broker.

The application receives the http request executes the business logic, communicate with other systems and database and return the response in differtn formats such as xml/json/text. The application can be logically splitted into above with respect to the different functional areas of the application.

**Solution:**

* For the above scenario, use the monolithic architecture to develop the application.
* In monolithic application, the application wrapped into single war file.

**Benefits or Advantages:**

* **Simple to develop :** it simple to develop with current IDE and development tools.
* **Simple to Deploy:** it is simple to deploy as single war
* **Simple to scale:** it is simple to scale horizontally by adding more machines or running the multiple copies of the application behind the load balancer.

**Issues and Limitations(disadvantages):**

* **Difficult to understand the Coding:** Large monolithic application intimidates(frighten) the developers. Being the new team member it is difficult to understand the codebase and do the modification.
* **Overloaded IDE:** large monolithic code base make the IDE slower in process . As a result the developers productivity drcreased.
* **Overloaded we container:** large monolithic application make the container slower in start up process. So the developer has to wait for the container to start the application. As a result developers productivity decreases and also it affects deployment.
* **Continuous deployment is difficult:** In large monolithic application, if there is change in one component , entire application needs to be deployed. In case the updated component is not started correctly , it forces the redeployment which discourages the continuous deployment. This is especially problem for UI developers.
* **Scaling the application can be difficult:** Monolithic architecture supports to scale the application on one dimension ie.horizantally. This is done by increasing the copy of the application on different machines on different machine. It doesn’t support the scaling in vertically by increasing the data volume. It doesnt not support to scale the compenents independently. Different components may need of different kind of resource like one may need CPU ,other may need memory , since all aomponeent are tightly coupled , all need to access all types of data.
* **Scaling the development is difficult:** there are various teams involved in the development of monolithic application such as UI team , inventory team, stock team etc. The team need to be coordinated each other on their development and deployment efforts. They cannot work independently.
* **Restriction on Technology:** In monolithic architecture ,once the technology is selected at starting the application development , adopting to new technology is very difficult. If we want to migrate to the new technology , the entire application to be rewritten which is risky.

To overcome these limitations we are moving to micro services architecture.

Consider the same scenario explained above.

**Solution: (in Microservices architecture)**

Define an architecture that structure the application s set of loosely coupled independent services. Each service has its own functionality. Each component is developed and deployment as independent service. Each service has its own database. The services are communicating each other by either synchronous protocols such as http/REST or asynchronous protocols such as MQ. This architecture supports the application to be scaled vertically by increasing the data volume and scale the component independently. Data consistency between services are maintained by event driven architecture.

**Benefits or Advantages: ( in microservices)**

* The microservices are small so
* It can be easily understanding by the new team members and be productive
* Since the it lesser code , IDE makes faster and increasing the productivity
* Since its lesser code , makes the container to start the service faster which speeds up the deployments
* It supports continuous deployment since each component deployed independently. If any change in one component , only that particular components needs to be deployed which needs to deploy the new version of that particular component
* It supports scaling the application vertically by increasing the data volume. The individual component may access the only the required data.
* If any issue in one service , only that particular service or component is affected. The other services can run independently.
* It can easily adopt to new technology. When developing the new service , we can pick the new technology for development.

**Disadvantages:**

* Developers must deal with the complexity of developing the distributed applications.
* Deployment Complexity. Since each service needs to be deployed separately
* Increased memory consumption. Each service runs on its own jvm , then memory consumption is increased.
* The IDE / tools used for developing the monolithic application is not sufficient
* Developers must implement the interservice communication mechanism.
* Developing the use cases that span across multiple service need team coordination.

**How to decompose the monolithic application into micro services?**

* **Decompose by business capability:**
* **Decompose by subdomain:**

**How to maintain the data consistency between microservices?**

Event driven architecture is use ful to maintain the data consistency between services. Each service has its win database. Whenever therei is a change in data in one service it generates an event , the other services consume that event and updated their data. There are several ways used in this architecture.

* Event Sourcing :
* Application events : The application inserts an event into event table. Another process pools the event and publish that events to message broker.
* Database triggers : database triggers are inserts the event in the events table , another process pulled that events and publish
* Transaction log tailing: Tail the database transaction log and publish each change as an event

**4) What is network threats ?**

There are 4 types of network threats,

1. Unstructured - virus,worm,Trojan horse
2. Structured - Identity theft , credit card information theft
3. Internal : created by the ex-employee of the organization
4. External: created by the person outside of the organization.

**5) When do we choose SOA Architecture ?**

**SOA –**Service Oriented Architecture.

It is an architecture used to integrate the different components exposed as service to build an application in a distributed environment. It also connects the systems in heterogeneous environment. It is also used for the reuse of the legacy application. It is also used when we need to abstract the implementation details to customer. Though we have the traditional distributed messaging techniques such as RMI and CORBA, it cannot connect the heterogeneous systems that are separated by firewalls. This is because firewalls allows only HTTP and SMTP traffic. This prevent the enterprise applications, to connect the distributed systems available over the LAN. The SOA architecture address this problem as exposing the systems as services instead of plain objects. The SOAP or Other XML messages are designed to communicate these systems.

**6) Why do we need web service security (WS-Security)?**

The component are exposed as service in SOA architecture and these services are available on over the internet to consume. The XML messages designed for the SOA Architecture to communicate with these services are firewall transparent and these messages can bypass the firewalls over HTTP and SMTP protocols. These kind of message transparency lead to security issues. So we need to use the WS-Security protocol to protect the messages from end to end(sender to receiver) in SOA system. But this WS-Security standards ensures the security between two trusted parties.

1. **What are the security vulnerabilities arising in Web applications?**

There are so many scenarios , the security vulnerability issues may happen in the web applications.

**Scenario 1: Input Validation attacks**

If the input validation not happened in the web application , there are so many possible causes which attacks the web application.

1. **Buffer Overflow attack :**

By inserting the larger value than expected into program variable may execute some arbitrary or malicious code with the privilege of root user. Each process has its own memory , so in this scenario the memory allocated for the process is smaller than the data written to the memory. So the extra data goes some where undesirable.

**Possible Cause :**

Size of the input is not validated before written to memory.

**Solution:**

1. Input size validation should be done for the program variable.
2. **Cross Site Scripting : ( XSS)**

Client side script injection. Injecting the client side script to the web pages. The attackers explore the web application and find the page which doesn’t filter out the input for the authenticated users. They will simply insert the malicious code or script as part of the request which will be stored in the database , visitors log, comment field or message forum. Later when the actual user try to access the page , this malicious code or script send as part of the output , which causes the client browser. The client browser doesn’t know that these are sent from untrusted sources. This kind of script can easily access the users password, credit card details, session tokens and other sensitive informations from the web application. This way the attackers attack the web application. Most of the XSS attacks happens by using the <Script> tags in the request.

**Types:**

1. **Stored XSS :** The user input is stored in the data base ,message forum, visitors log and send to client browser as part of the output
2. **Reflected XSS :** The user input is immediately returned to the client as part the the response.

**Possible Cause:**

Input and output are not validated and not encoded properly.

**Solution:**

Input from user and output send to browser needs to be validated and encoded properly. <Script> tags needs to be filtered out in the input and output.

1. **SQL Injection:**

Explained earlier.

**Possible Cause:**

Unvalidated User Input

**Solution:**

User input needs to be validated. Special characters needs to be filtered out in appropriate places. Single and double quotes needs to be removed from input.

1. **IP-spoofing:**

Computers network communicate each other by exchanging the information in form of data packets. These packets contain various header information. One such header is “Source IP Address”. IP-Spoofing is also known as IP address forgery or host file hijack. This is the hijacking technique used by the attacker who create the fake IP packets and sent it to the clients. Once the attacker gains access to the network they can modify the data, reroute or delete the data. Sometimes they can steel the sensitive information from client system or they can install malicious code into the client systems.

**Possible Cause:**

Using the direct IP address authentication .

**Solution :**

Don’t use direct IP address based authentication.

Use some access control list in downstream interface to deny private IP address.

Filter the data packets on the router which comes from outside the network

Use public key based authentication.

Enable encryption based session , so that trusted sites outside of the network can communicate with actual hosts.

1. **Network eavesdropping:**

Eavesdrop :- secretly listens to a conversation.

This vulnerability occurs when the attackers gains access to the data path to the network. This can be done by IP- Spoofing or by Man-in the-middle attacks.They can easily capture the user name and password which send from client to server in form of plain text. By using these details they can easily capture the sensitive data by eavesdropping the connection. The victim(client) doesn’t know that he is monitored by the attacker or the presence of the attacker in the middle.

1. **Dictionary Attack:**

The attackers goal is to obtain the password. So they systematically test all the password until they found the correct one.

**Possible Cause:**

Passwords are sent as part of request which is not encrypted

**Solution :**

Use proper encryption method for the inputs and outputs.

1. **Data Tampering:**

**Tampering :-** interfere with something to cause damage or perform unauthorized alteration.

This is occurs when the attackers gain access to the data passes over the network . The attacker can manipulate the data sent from client to server. As a result the tampered data sent to the origination.

**Possible Cause:**

IP-Spoofing

**Solution:**

Prevent the ip-spoofing

**Scenerio 2: Denial Of Service attack**

The main goal of the attackers is to crash the web application by stealing secret information. With this attack they can easily crash the providers computer or network. They can even crash the router , firewall or proxy servers.

**Possible Cause:**

IP-Spoofing

**Solution:**

Prevent the ip-spoofing

**Scenerio 3: Session Management attacks**

Weak session management results in session management attacks in web application. This attacks occurs when the attackers gains access to the session ID such as cookie,url parameter , session id.

1. **Session Replay:**

This is the hijacking technique used by the attacker to perform the session replay. They steel the session id’s which is sent from client to server and replay the same request by bypassing the authentication. So that they get the system access.

1. **Man-in-the-Middle:**

This occurs when the attacker intercepts the message between client and server. They can easily modify the data and send to sender and receiver . The sender and receiver doesn’t aware of the Main in the middle.

**Scenerio 4: Parameter Tampering**

The goal is to attack he form parameters when it is send from client to server. The attacker can manipulate the header parameter and mode parameter. The parameter manipulation has the following types,

1. **Http header manipulation:**

The header always has the control information. Http request header is sent from client and Http Response header is send from server. Web browsers usually doesn’t allow to modify the headers. But the attackers can write their own program to manipulate the headers.

1. **Form Field manipulation:**

The attacker can change the form field values such as hidden filed, free form field ,check box, option box by page- . view source->save -> modify the content -> reload the page.

1. **QueryString manipulation**

When the client use Http Get method to send the request , the request parameters are appended with the url and displayed I the browser. The attacker can easily change this data.

**8) What are the security vulnerability arise in web services? How to prevent this?**

Apart from the above web application vulnerability, there are some more applicable for the XML based web services.

Because of the open standard and modularity nature of the web services still lead to security problems. SSL itself is not sufficient for the web service security, since SSL is not designed for the internet based web services. SOAP also doesn’t have its own security mechanism. Neither firewalls nor PKI(Public Key Infrastructure ) based security mechanism helpful for web service security. Because firewalls allows HTTP traffic via web service request. PKI security mechanisms is also not sufficient for the chains of applications connected by web services. WSDL also show the data elements very clear Attackers always interested on the XML message for their attack.

**XPath Injection:**

This is also an vulnerability arise in XML based web application. Similar to SQL injection. When we use the user input directly to the XPath queries , all the data can be fetched from the Native XML document in which application uses XML as data store.

Application stores employee details in XML store,

<?xml version="1.0" encoding="utf-8"?>

<employees>

<employee id="AS789" firstname="John" lastname="Doo" annualsalary="70000"/>

<employee id="AS719" firstname="Isabela" lastname="Dobora" annualsalary="90000"/>

<employee id="AS219" firstname="Eric" lastname="Lambert" annualsalary="65000"/>

</employees>

XPath Injection happens like this,

Document doc = builder.parse(new InputSource(new StringReader(DATASOURCE\_XML)));

// Retrieve employee ID from the input HTTP request

String eID = request.getParameter("employeeID");

if (eID == null) {

eID = "";

}

// Create XPATH expression

String xpathExpr = "/employees/employee[@id='" + eID + "']";

XPath expression = new DOMXPath(xpathExpr);

The attacker can change the request input and can fetch the irrelevant data from xml store.

By using the this XPath queries they can easily extract the complete XML document. XPath is also useful to identify the parts of the XML to which the XML signatures applies.

**SOAP Security:**

SOAP doesn’t define any security mechanism for the web services. SOAP message can easily pass through the firewalls using the HTTP protocol. SOAP messages are represented as SOAPEnvelope , and it has two element as SOAPHeader and SOAPBody. SOAP Header is an optional element and it contains the security and other soap message level details. It can also contains some details about SOAP server. The attacker use this server details and they guess the endpoint details and they can route the messages to some other destination by changing the header details. Head is an optional element. SOAP message can have one or more header element. Security element in header section specify the target of the role. If no security element present , it can be consumed by anyone. The attacker can insert extra header or they can delete the header or they can delete the body of the soap message.

To prevent this , SOAP messages needs to be encrypted and digital signature to be used.

**XML Security:**

XML encryption and XML digital signatures are two methods used to secure the XML messages. There are different algorithms are used in these techniques. Only using these techniques doesn’t make the web services secured. The other security attacks and solutions are described below,

**XML digital signatures:-** provides authentication, data integrity and non reputation. Encrypt the whole document using the digital certificate

**XML Encryption:-** provides secure XML document. It encrypt only part of the XML document.

**Classification on XML Web Service Security attacks and Solution :**

**Refer the table 1 from WebServiceSecurityPartII.pdf**

**9) How the webservice is digitally signed ?**

**Refer :** [**http://www.drdobbs.com/database/digitally-signing-and-verifying-messages/209400693?pgno=2**](http://www.drdobbs.com/database/digitally-signing-and-verifying-messages/209400693?pgno=2)