Argentina $8,90 (recargo al interior $0,20) / México: $45

USERS

Microsoft

Curso teórico y práctico de programación

Desarrollador .net

Con toda la potencia de **Visual Basic .NET** y **C#**

La mejor forma de aprender a programar desde cero

★★★★★

Basado en el programa Desarrollador Cinco Estrellas de Microsoft

17

SALIDA LABORAL
La mejor preparación para rendir los exámenes
GARANTIZADA

Seguridad

Plataforma de protección para componentes, bases de datos y servicios

Framework 3.0

Todas las novedades - WPF, WCF y WF

ISBN 978-987-1347-43-8

9 789871 347438 00017

RedUSERS

COMUNIDAD DE TECNOLOGIA

¡SUMATE!

EL SITIO Nº1 DE TECNOLOGIA

Noticias al instante // Entrevistas y coberturas exclusivas // Análisis y opinión de los máximos referentes // Reviews de productos // Trucos para mejorar la productividad // Registrate, participa, y comparte tus opiniones

SUSCRIBITE

SIN CARGO A CUALQUIERA DE NUESTROS **NEWSLETTERS** Y RECIBÍ EN TU CORREO ELECTRÓNICO TODA LA INFORMACIÓN DEL UNIVERSO TECNOLÓGICO ACTUALIZADA **AL INSTANTE**

INGRESÁ A
redusers.com/suscribirse-al-newsletter
¡Y REGISTRATE YA!

Foros · Encuestas · Tutoriales · Agenda de eventos · Videos · ¡Y mucho más!

redusers.com

Seguinos en:
www.facebook.com/redusers
www.twitter.com/redusers
www.youtube.com/redusersvideos

Consumir un Web Service

Revisaremos las cuestiones para tener en cuenta al momento de consumir un servicio Web que hayamos construido.

La finalidad de crear un servicio Web es consumirlo o utilizarlo. Esto se puede realizar tanto desde una aplicación Web como desde una página Web. El resultado obtenido con una llamada al método requerido debería de ser el mismo para ambos casos. También existe una tercera posibilidad, que es consumirlo a través de otro servicio Web que, a su vez, es consumido por una aplicación Web o Windows.
Vamos a comenzar por crear una aplicación Windows, para lo cual generamos un nuevo proyecto al que denominaremos winAritmetica. Luego de hacerlo, como primera medida, nos dirigimos a las referencias y, con el botón derecho, seleccionamos la opción de agregar referencia Web. Ésta abre una ventana de exploración de servicios Web, donde podemos colocar directamente la URL del servicio deseado o bien escoger ir a los servicios Web del equipo local. También podemos explorar los servicios Web que estén publicados dentro de la red interna y otras opciones más.
En nuestro caso, optamos por visualizar los servicios del equipo local; aparece una lista de todos los que están publicados en nuestra PC. Hacemos clic en el deseado, en nuestro proyecto, wsAritmetica. A continuación, pasamos a una pantalla donde se visualiza el servicio como si lo navegáramos desde un explorador de Internet. A la derecha vemos una casilla de texto donde debemos colocar el nombre que queremos darle a nuestra referencia Web; en nuestro caso, es wsAritmetica, para no crear confusiones. Por último, presionamos el botón Agregar referencia.

El resultado de estas acciones se puede observar en el explorador de soluciones de nuestro proyecto, donde veremos una carpeta nueva denominada Referencias web, con todas las que hayamos incluido (en nuestro caso, una sola, denominada wsAritmetica).

Al formulario de inicio se le han agregado dos cuadros de texto llamados txtNumero1 y txtNumero2, con el fin de que contengan los números que se van a operar mediante el servicio Web. También hay tres botones llamados bSuma, bResta y bDividir, y cuyos textos fueron establecidos como Sumar, Restar y Dividir, respectivamente. Por último, se agregó un cuadro de texto de sólo lectura denominado txtResultado, que será el encargado de mostrar la respuesta devuelta por el servicio Web.
Por otro lado, se declaró una variable del tipo wsAritmetica.aritmetica, que es un objeto del tipo de nuestro Web Service, y en el evento clic de cada botón se invocó al método correspondiente de nuestro servicio y se muestra el resultado en el cuadro de texto destinado a tal

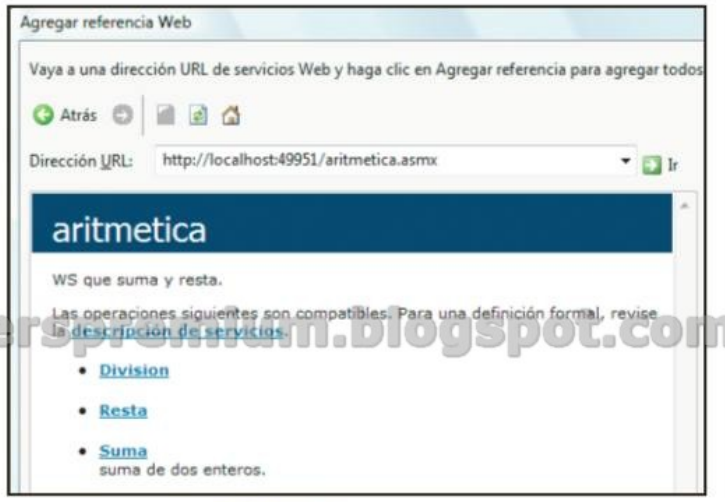

FIGURA 012 | Ventana que nos permite agregar nuestra referencia Web.

La finalidad de crear un servicio Web es consumirlo o utilizarlo. Esto se puede realizar tanto desde una aplicación Web como desde una página Web.

efecto, siempre realizando las conversiones de tipo correspondientes a cada caso.
El código agregado es el siguiente:

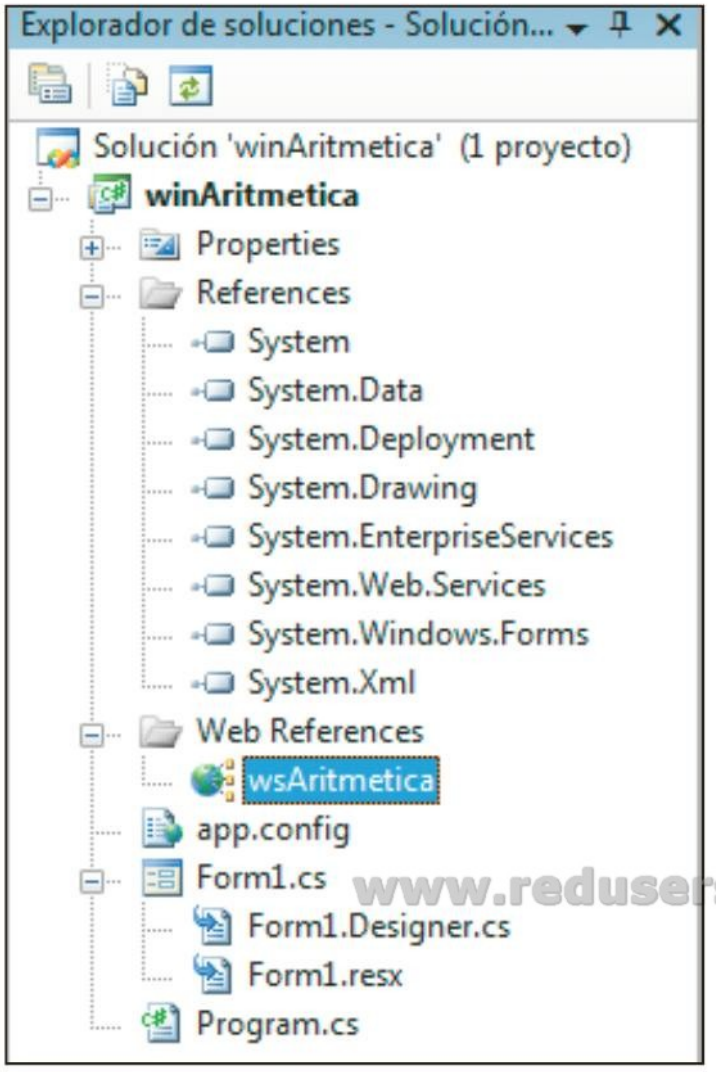

FIGURA 013 | Aquí se observa la nueva carpeta de referencias Web.

```
...
 private wsAritmetica.aritmetica servicio =
 new wsAritmetica.aritmetica();

 private void bSumar_Click(object sender,
 System.EventArgs e)
 {
  txtResultado.Text = (servicio.Suma(Convert.
  ToInt32(txtNumero1.Text), Convert.ToInt32
  (txtNumero2.Text))).ToString();
 }

 private void bRestar_Click(object sender,
 System.EventArgs e)
 {
  txtResultado.Text = (servicio.Resta
  (Convert.ToInt32(txtNumero1.Text), Convert.
  ToInt32(txtNumero2.Text))).ToString();
 }

 private void bDividir_Click(object sender,
 System.EventArgs e)
 {
  txtResultado.Text = (servicio.Division
  (Convert.ToInt32(txtNumero1.Text), Convert.
  ToInt32(txtNumero2.Text))).ToString();
 }
...
```

En este momento, ya estamos en condiciones de ejecutar nuestro proyecto Windows Form y verificar el comportamiento de nuestro servicio Web. Como podemos observar, al declarar el objeto que se encarga de exponer los métodos del servicio Web, lo hacemos bajo el nombre de referencia Web que hemos colocado cuando generamos el enlace. Luego, éste nos expone todas las clases que pueden ser instanciadas y, por último, cada una nos permite invocar a cada uno de los métodos Web que fueron codificados al momento de crear el Web Service. Podemos notar que no hay nada de complicado, al

menos hasta aquí, y que en forma Web sería algo similar a lo que podemos observar a continuación.
Creamos un proyecto del tipo ASP.NET al que denominamos webAritmetica, en contraposición al anterior. De manera análoga a la construcción del proyecto previo, agregamos la referencia a nuestro servicio Web y, con el fin de evitar diferentes denominaciones, la llamamos wsAritmetica.
Para el caso de este proyecto de tipo Web, se ha elegido agregar dos cuadros de texto: **txtNumero** y **txtNumero2**; un botón denominado **bOperar**, cuyo texto será **Operar**; y tres cuadros de texto: **txtSuma**, **txtResta** y **txtDivision**, que contendrán los resultados de las operaciones. En este caso, el botón llamará uno a uno a los tres métodos expuestos por nuestro servicio Web y mostrará los resultados en los cuadros correspondientes.

También agregamos la declaración del objeto y los métodos dentro del evento clic del botón, con lo cual finaliza el proyecto:

```
...
 private wsAritmetica.aritmetica servicio =
 new webAritmetica.wsAritmetica.aritmetica();

 private void bOperar_Click(object sender,
 System.EventArgs e)
 {
  txtSuma.Text = (servicio.Suma(Convert.
  ToInt32(txtNumero1.Text), Convert.ToInt32
  (txtNumero2.Text))).ToString();
  txtResta.Text = (servicio.Resta(Convert.
  ToInt32(txtNumero1.Text), Convert.ToInt32
  (txtNumero2.Text))).ToString();
  txtDivision.Text = (servicio.Division
  (Convert.ToInt32(txtNumero1.Text), Convert.
  ToInt32(txtNumero2.Text))).ToString();
 }
...
```

Un método Web es tratado de forma similar a como se trataría cualquier método de una clase común y corriente.

Para sumergirnos más en el significado de un servicio Web, podemos observar que también expone otros métodos, como **BeginSuma** y **EndSuma**, los cuales son generados automáticamente e invocados por el propio servicio en forma automática antes y después, respectivamente, de la llamada al método Web denominado Suma.

Veamos otra manera más de consumir un servicio Web. En este caso, creamos un nuevo proyecto del tipo servicio Web, con la finalidad de generar un nuevo servicio al que llamaremos **ws2**. Una vez hecho esto, agregamos **wsAritmética** como referencia Web, como ya explicamos anteriormente.
Lo que se pretende con este nuevo servicio es demostrar cómo se puede consumir un Web Service desde otro. Por ejemplo, en el caso anterior contábamos con métodos Web que suman, restan y dividen, pero no que multiplican. Para esta última operación, usaremos un

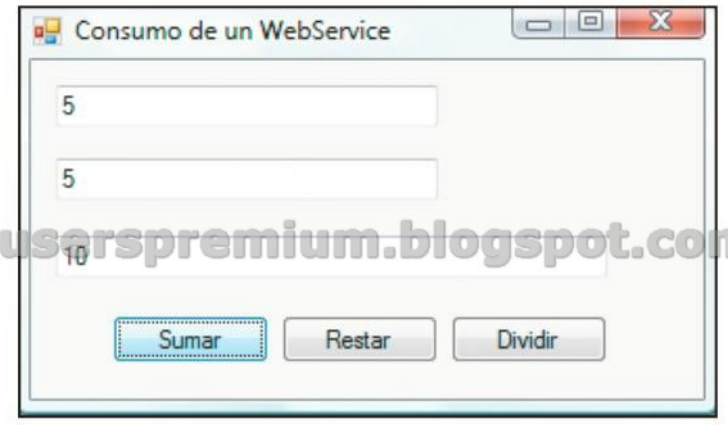

FIGURA 014 | Nuestro ejemplo en plena ejecución.

Un servicio Web también expone otros métodos, como BeginSuma y EndSuma, los cuales son generados automáticamente.

nuevo servicio Web, al que le crearemos un nuevo método Web denominado multiplicación. A diferencia del anterior, haremos de cuenta que nuestra PC no cuenta con la posibilidad de multiplicar, sino que la implementaremos a través de sumas, invocando al método Web Suma de nuestro primer servicio. El código fuente será el siguiente:

```
...
 [WebMethod]
 public int Multiplicacion(int a, int b)
 {
  int resultado = 0;
  wsAritmetica.aritmetica arit = new
  wsAritmetica.aritmetica();
  for (int i = 1; i <= a; i++)
  {
   resultado = arit.Suma(resultado, b);
  }
  return resultado;
 }
...
```

FIGURA 015 | Se observa aquí la navegación de nuestro método Web multiplicación.

Expliquemos brevemente este método con el fin de que sea comprendido por los menos aficionados a las ciencias exactas. Se ingresan dos números denominados factores (a y b) y se inicializa una variable resultado en 0 (si vamos a realizar varias sumas sobre esa variable, debe ser inicializada con el elemento neutro de dicha operación). Por otro lado, declaramos un objeto **arit**, que hace referencia a nuestro primer servicio Web. Luego entramos en un ciclo "a" veces, en el que acumulamos en "resultado" la suma de "resultado" y "b". Para sumar no utilizamos la suma de nuestra computadora, sino el método Web **Suma** de nuestro primer Web Service.
A continuación, podemos navegar nuestro servicio Web y si, por ejemplo, colocamos los valores 3 y 5, obtendremos como resultado 15.

Otra cosa que puede resultar de importancia a la hora de construir y consumir un servicio Web es el sistema de autenticación o bien el modo de conservar valores dentro de la sesión. Mostraremos a continuación el código de un servicio Web con dos métodos denominados **Inicio** y **Saludo**. El primero recibe un parámetro del tipo **string**, donde colocaremos un nombre; y el segundo no recibe nada, pero nos devuelve un saludo. Cabe aclarar que para este ejemplo se presupone que las llamadas se hacen desde la misma instancia (por ejemplo, sin cerrar nuestro explorador) y en el orden precisado, ya que no es el propósito de este capítulo mostrar cómo se trabaja con sesiones.

```
...
 [WebMethod(EnableSession = true)]
 public void Iniciar(string nombre)
 {
  Session["nombre"] = nombre;
 }

 [WebMethod(EnableSession = true)]
```

```
public string Saludo()
{
 return "Hola: " + Session["nombre"];
}
...
```

Los agregados que podemos encontrar son unos atributos de método que, en este caso. es **EnableSession** y que se ha establecido en **true**; esto hace que entre una llamada y otra se haya mantenido el estado de sesión. Entonces, nuestro primer método (**Iniciar**) almacena el valor ingresado dentro de una variable de sesión, en tanto que el segundo (**Saludo**) lo recupera y lo devuelve dentro de un **string** precedido por un **hola**.
Si consumiéramos este servicio Web desde nuestro navegador, al invocar el método Iniciar, obtendríamos como resultado una página en blanco, y al hacerlo con el método Web Saludo, obtendríamos un tag XML con el tipo de dato y el mensaje de saludo, como se puede ver a continuación:

```
<string>Hola: Emilio</string>
```

Como podemos notar, el nombre ingresado en la invocación del método Iniciar fue Emilio. Entre estos atributos, ya hemos usado **Description** y **EnableSession**. También hay otro denominado **MessageName**, que establece un nombre paraa nuestro método Web. Esto es práctico a la hora de tener **WebMethods sobrecargados**. Los métodos Web no pueden sobrecargarse, así que si queremos construir dos métodos con el mismo nombre y que reciban diferentes parámetros, debemos especificar este atributo que, sí o sí, tiene que ser diferente entre todos los métodos Web de nuestro servicio.

También podemos crear clases dentro de nuestro servicio Web para que, de alguna manera, puedan ser consumidas dentro de él y, por supuesto, utilizadas por nuestros clientes, tanto Web como Windows. Para demostrar esto, hemos creado una clase denominada Persona, que se construye sobre la base de dos parámetros (apellido y nombre) y que posee un único método público sobrescrito, bien conocido por todos: **ToString()**. Dentro del servicio Web anterior hemos agregado otro método Web saludo, que implementa esta clase como parámetro. El código de nuestro servicio Web es el siguiente:

> Los métodos Web no pueden ser sobrecargados, pero sí pueden llamarse igual si se setea un atributo que le otorgue un alias.

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

namespace wsSession {

 /// <summary>
 /// Descripción breve de wsSession
 /// </summary>
 [WebService(Namespace = "http://sitio.com/")]
 [ToolboxItem(false)]
 public class wsSession : System.Web.
 Services.WebService {

  [WebMethod(EnableSession = true)]
  public void Iniciar(string nombre) {
```

```
    Session["nombre"] = nombre;
  }

  [WebMethod(EnableSession = true,
  MessageName = "SaludoComun")]
  public string Saludo() {
    return "Hola: " + Session["nombre"];
  }
  [WebMethod(EnableSession = true,
  MessageName = "SaludoObjeto")]
  public string Saludo(Persona p) {
    return p.ToString();

  }

}

public class Persona {

  public Persona() {
  }

  public Persona(string apellido, string
  nombre) {
    ap = apellido;
    nom = nombre;
  }

  private string ap = String.Empty;
  private string nom = String.Empty;

  public override string ToString() {
    return ap + ", " + nom;
  }

}
```

Podemos observar como primera medida la clase Persona; sólo se coloca el código con fines de consumir esta clase.

A la hora de construir un servicio Web, es importante el modo de conservar valores dentro de la sesión.

Vemos que en nuestro servicio han quedado dos métodos con el mismo nombre (Saludo), por lo que hubo que colocar el atributo **MessageName** con el fin de declarar un alias diferente para cada uno, según como vimos anteriormente. Nuestro nuevo método recibe una instancia de la clase Persona y devuelve lo que el método **ToString** de la clase Persona le retorna, o sea, el apellido seguido del nombre, con una coma y un espacio que los separa.

Para consumir este nuevo método se debe hacer lo siguiente:

```
...
private wsSession ws = new wsSession();

private ws.Persona per = new ws.Persona
("Pérez", "José");

labelPersona.Text = ws.Saludo(per);
...
```

En este ejemplo, hemos creado un objeto del tipo de nuestro servicio Web; luego, una clase del tipo persona, y lo construimos a través de los dos parámetros requeridos por el constructor; por último, cargamos en la propiedad **Text** de un **Label** la respuesta de nuestro servicio. Podemos observar que es como trabajar con una referencia común dentro de nuestro proyecto, creando objetos declarados fuera de él (dentro de la referencia), utilizando métodos, etc.

Seguridad

Implementar un desarrollo de software seguro

10

Contenidos

En este capítulo aprenderemos los conceptos básicos sobre la seguridad en aplicaciones, sus tipos y las amenazas más conocidas para aplicaciones Web. También veremos cómo construir servicios y componentes, y diseñar bases de datos seguras.

Temas tratados

- Introducción
- Amenazas en aplicaciones Web
- S.T.R.I.D.E.
- D.R.E.A.D.
- Ataques
- Plataformas de protección
- Seguridad de componentes
- Seguridad de bases de datos
- Seguridad de servicios

Seguridad

Conoceremos las técnicas utilizadas para identificar distintos tipos de amenazas y los ataques más comunes. También aprenderemos a construir desarrollos seguros.

Introducción

Aprenderemos los conceptos básicos sobre seguridad de software, cómo se aplican controles físicos y lógicos de seguridad, y las consideraciones que debemos tener para garantizar el resguardo.

- Problemas de seguridad
- Concepto de seguridad
- Producir software seguro

Amenazas en aplicaciones Web

Las aplicaciones Web son más propensas a estar bajo amenaza. Aquí aprenderemos cómo identificar estos riesgos y de qué manera tratarlos; incluso, cómo idear acciones para prevenir estos peligros.

- Objetivos comunes de un ataque
- Fallas en la codificación
- Scripting
- Modelado de amenazas

S.T.R.I.D.E.

Aprenderemos acerca de esta técnica de modelado de amenazas, para poder crear un plan de pruebas sobre nuestra aplicación.

- Objetivo de esta técnica
- Significado de las siglas

D.R.E.A.D.

Veremos que, una vez identificadas las amenazas, es fácil darse cuenta de que no todas tienen probabilidades de hacerse efectivas.

- Significado de las siglas
- Objetivo de cuantificar cada amenaza
- Análisis de amenazas aplicando D.R.E.A.D.

Ataques

Las aplicaciones Web son las más propensas a ataques, por lo que aprenderemos las formas que existen para contrarrestarlos.

- El scripting
- La suplantación
- Objetos embebidos
- SQL Injection

Seguridad

En las siguientes páginas veremos cuáles son los pasos por seguir para brindar seguridad en forma integral.

La seguridad en el desarrollo de software suele estar asociada a la idea de cuán propenso puede ser nuestro producto o líneas de código a posibles ataques externos, o sea, fuera del dominio de la aplicación. Si bien éste puede ser un concepto relativamente acertado, en el proceso de desarrollo de software, las líneas de código no son los únicos factores involucrados.
En el transcurso de este capítulo veremos cómo los problemas de seguridad pueden ser disparados tanto por la escritura de líneas de código problemáticas, como por el humano, y de qué manera atenuar y contrarrestar estos inconvenientes.

De cualquier modo, la seguridad en el software está atada no sólo a esta capacidad de interconexión, sino también a la capacidad de un producto de cumplir con lo esperado: un sistema operativo, a mantener correctamente el sistema de archivos o impedir acciones restringidas sin aprobación por parte de su propietario; una aplicación Web, que no exponga datos críticos (conexión a la base de datos, nombre de usuario del administrador, etc.) con sólo colocar un par de líneas de código básicas en cualquier campo de texto en una de sus interfaces.

Por otro lado, para poder producir nuestro software, éste debió haber pasado por un proceso de producción, y este proceso tiene un alcance mayor que la simple escritura de líneas de código. Como ejemplos podemos nombrar los contactos con el o los clientes que pueden encargar la producción del software, los arquitectos que diseñen la estructura de las clases, un agente externo que nos provea del hardware necesario para la puesta en marcha de las pruebas, hasta los diferentes departamentos de nuestra empresa. Todos estos actores poseen, de una u otra forma, acceso a información crítica de nuestro proyecto, y pueden divulgarla, con lo cual causarán un daño al producto, tanto como un código mal creado.

La seguridad en el software está atada no sólo a la capacidad de interconexión, sino también a la capacidad de un producto de cumplir con lo esperado.

Por este motivo, es común encontrarse con controles de seguridad en las empresas, a nivel tanto

! El concepto de seguridad

Si bien el concepto de seguridad sobre software existe desde el mismo principio de la informática, ha cobrado mayor notoriedad (o visibilidad) en la última década y media, en gran medida, gracias a la capacidad de exponer nuestras computadoras a otras cientos de miles en Internet, redes inalámbricas, Bluetooth, etc.

lógico como físico. Dentro de los controles lógicos más comunes podemos nombrar:

- Acceso restringido a páginas en Internet por parte de los empleados.
- Antivirus corporativos.
- Instalación de parches de seguridad.
- Instalación de software por parte del usuario anulado. La cuenta de usuario (nombre de usuario y contraseña) que usa para ingresar a su equipo no posee privilegios para instalar aplicaciones.
- Firma de contratos de confidencialidad por parte de los empleados y de los proveedores.

Como controles físicos:

- Bloqueo de puertos USB en las terminales de trabajo.
- Uso de credenciales de activación de puertas para el ingreso al área de trabajo.
- Toma de números de serie de equipos que se ingresen a la empresa, y verificación de éstos en el momento en que salen.

Como hemos visto, la seguridad no es un concepto relacionado sólo con el software, sino que representa una cuestión mayor que involucra todo el proceso de desarrollo. Entonces, en las siguientes páginas veremos cómo planificar, ejecutar y contrarrestar las acciones que pueden dañar nuestros productos. En definitiva, aprenderemos a pensar en seguridad desde la concepción misma de la idea de nuestro producto.

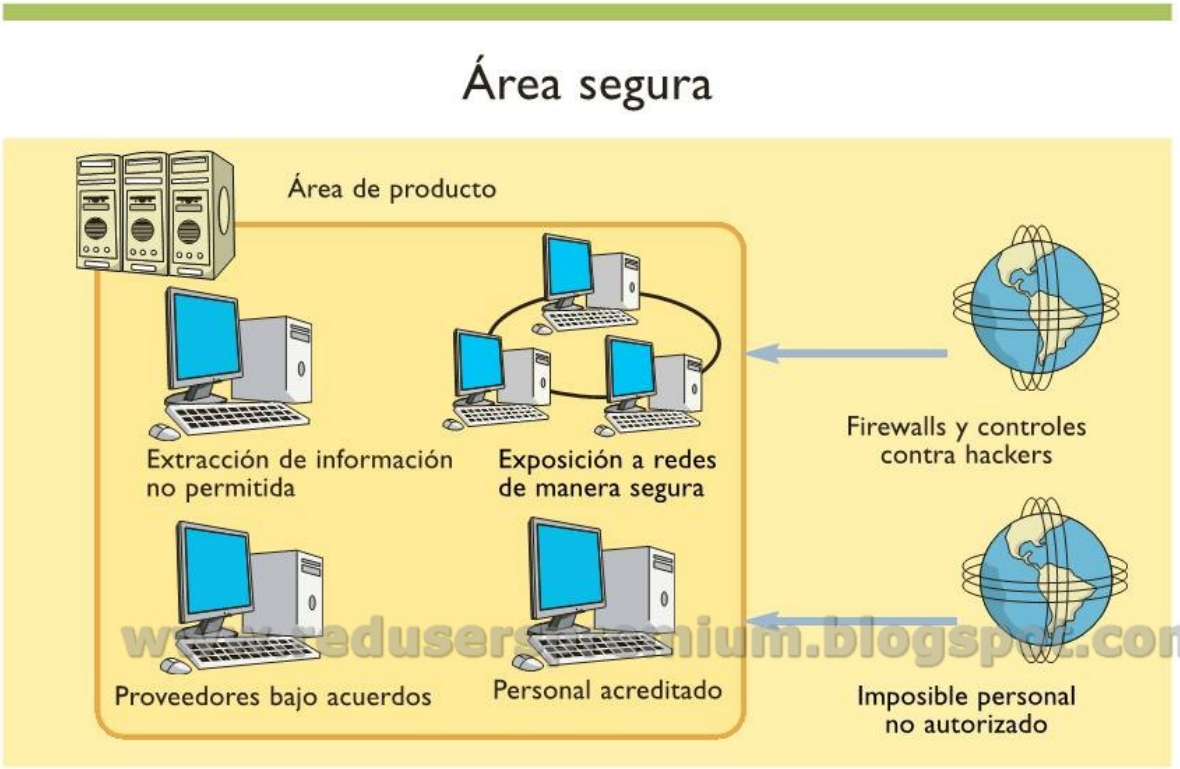

FIGURA 001 | Consideraciones logísticas que dan seguridad en un ambiente de desarrollo.

Amenazas en aplicaciones Web

En las siguientes páginas veremos cuáles son las amenazas más comunes que pueden sufrir nuestras aplicaciones Web.

Como vimos anteriormente, cuando nuestros equipos comienzan a conectarse a redes compartidas con miles de usuarios, es el momento en que la seguridad en las aplicaciones se torna una realidad inevitable.
Las aplicaciones Web, debido al ámbito en el cual trabajan, son propensas, en una escala mayor que otras, a estar bajo amenazas que pueden afectar su funcionamiento.
Debemos tener en cuenta que una aplicación Web es un programa expuesto a cientos de miles de usuarios a través de Internet. Éste, ideado para satisfacer diferentes necesidades, podría almacenar información valiosa de aquellos que hacen uso de él, como ser:

- Cuentas de correo electrónico
- Números de tarjetas de crédito
- Datos financieros
- Direcciones personales, teléfonos, etc.

Esta información podría cumplir una tarea determinada para nuestra aplicación sin un motivo más allá de brindar un servicio (por ejemplo, una agenda electrónica accesible desde Internet para empleados de nuestra compañía) y sólo sería útil para nosotros. Pero en manos equivocadas podría ser usada para distintas técnicas ilícitas (ver Tabla 1).

En todo caso, cualquiera sea el objetivo perseguido por el atacante, los resultados nos comprometerán tanto como individuos como si somos parte de una empresa.

Amenazas por fallas de codificación

Si bien hasta el momento hemos mencionado fallas de seguridad, las vimos de manera general,

Tabla 1 | Objetivos más comunes en el ataque

Técnicas	Objetivo/Consecuencia
Robo de cuentas de correo electrónico	Envío de spam con publicidades no deseadas o virus que usarán el equipo infectado como robot para mandar más virus a otros usuarios.
Robo de números de tarjetas de crédito	En general, estos números se utilizan para realizar compras en otros sitios Web. El damnificado se hace cargo del pago por el robo de su tarjeta de crédito.
Robo de contraseñas	Se usan para acceder al sitio del cual se han robado estos datos, con el fin de obtener más información de la víctima y, así, poder aplicar algunas de las técnicas ya mencionadas.
Robo de propiedad intelectual	Se accede al servidor de la aplicación Web con el objetivo de robar propiedad intelectual. En algunos casos, también, se relaciona con el espionaje industrial.
Inhabilitación de servidores	Ataque ideado para reducir la capacidad de un servidor y/o una aplicación para brindar sus servicios. Causa pérdidas y retrasos en las tareas cotidianas del negocio.

> Como desarrolladores, es normal creer que el usuario de nuestra aplicación la usará como nosotros pensamos que debería hacerlo.

basándonos en las intenciones u objetivos perseguidos por el atacante.
Pero existen otras amenazas inherentes al código de nuestras aplicaciones, y el atacante tratará de valerse de fallas introducidas por nosotros en el momento de crear el código de nuestra aplicación.
Como desarrolladores, es normal creer que el usuario de nuestra aplicación la usará como nosotros pensamos que debería hacerlo. Por el contrario, es preciso tener en cuenta que el camino o flujo de la aplicación que hemos diseñado no es, muchas veces, el único posible.

Un caso común es el denominado scripting, que consiste en colocar código JavaScript en las entradas de texto de las páginas Web, con el objetivo de obtener información confidencial que pueda usarse en contra de la misma aplicación.
El siguiente código, que en circunstancias ideales mostraría el texto introducido por el usuario, podría ser potencialmente peligroso si no se aplican los controles necesarios:

VB.NET:

```
Protected Sub Button1_Click(ByVal sender As
Object, ByVal e As System.EventArgs) Handles
Button1.Click
   Response.Write(Me.TextBox1.Text)
End Sub
```

C#:

```
protected void Button1_Click(object sender,
EventArgs e)
{
   Response.Write(this.TextBox1.Text);
}
```

Una vez colocado el código JavaScript, éste es escrito en la página Web. El navegador lo tomará como código válido y lo ejecutará.

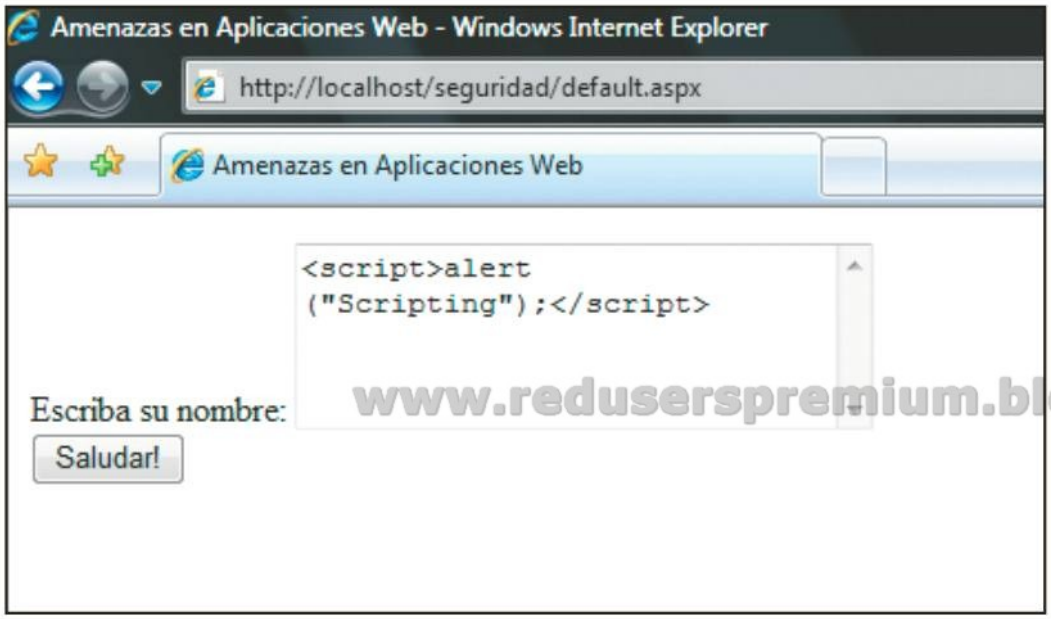

FIGURA 002 | En vez de introducir un texto, se hace uso de JavaScript.

Este ejemplo, por su simpleza, no muestra los peligros reales que tiene la técnica de scripting. Pero en casos más complejos, podría ser usada para el robo de contraseñas o datos críticos. Éste y otros peligros serán tratados en detalle más adelante.

> Si bien las amenazas son muchas, la mejor forma de contrarrestarlas es pensar como atacantes.

Modelado de amenazas

Si bien las amenazas son muchas, la mejor forma de contrarrestarlas es pensar como atacantes. Debemos ponernos en posición del agresor, probando todas las posibles fallas y analizando exhaustivamente nuestro código. Si detectamos un problema, aunque sea muy difícil de reproducir, alguien más podrá hacerlo, y explotará esa falla a su favor.

Esto puede lograrse siguiendo los pasos listados en la Tabla 2. El objetivo es ayudarnos a entender mejor nuestra aplicación, así como encontrar fallas en el código y en la arquitectura, y crear un plan de seguridad contra posibles ataques.

FIGURA 003 | Como resultado, se ejecuta el JavaScript y se muestra el mensaje.

Tabla 2 | Método de modelado de amenazas

Identificación de activos	Identificación de elementos en posible riesgo (servidores, bases de datos, páginas Web, etc.).
Visión de la arquitectura	Identificar qué hace la aplicación. Saber qué tecnologías están involucradas y documentar.
Descomponer la aplicación	Identificar los límites de confianza, el flujo de los datos y los puntos de entrada.
Identificar amenazas	Identificar las posibles amenazas para cada sección, usando S.T.R.I.D.E. o árboles de ataque.
Documentar amenazas	Llevar una bitácora de estas amenazas y de cómo contrarrestarlas.
Cuantificar amenazas	Otorgar una valoración del posible impacto y daño de la amenaza mediante D.R.E.A.D.

S.T.R.I.D.E.

Conoceremos una técnica utilizada para identificar amenazas según su categorización.

Este método tiene como objetivo categorizar cada amenaza dependiendo de su tipo. Por lo tanto, cada letra, por sus siglas en ingés, se ajusta a una categoría específica, que se describen en la Tabla 3.

Esta categorización es usada sólo con ese fin, y no tiene el objetivo de puntuar cada nivel. Es decir, todos deben ser tratados con el mismo cuidado e importancia. De cualquier manera, dependiendo de la aplicación, es probable que no todas sean aplicables.

Una forma práctica de aplicar esta técnica en un desarrollo sería analizar cómo podrían afectar estas amenazas a cada componente, a cada una de sus conexiones y/o respectivas relaciones. Tengamos en cuenta que, en algún momento, nuestra aplicación Web va a sufrir ataques, y siempre que un hacker pueda alcanzarla –es decir, que navegue por ella–, intentará hacer algo que no corresponde o algo malintencionado. En la mayoría de los servidores Web de acceso público, como Hotmail.com, de 4 a 7 veces por día se trata de encontrar alguna vulnerabilidad de la aplicación.

S.T.R.I.D.E. es una de las técnicas usadas en el modelado de amenazas para identificarlas.

Tabla 3 | Siglas de S.T.R.I.D.E.

Spoofing (falsificación)	Se refiere a la capacidad de la amenaza de falsificar la identidad de una persona. Es usada comúnmente en el spam, o por virus que leen la libreta de direcciones de la víctima para enviar una copia de éste con un nombre conocido para parecer confiable.
Tampering (manipulación)	Capacidad que posee una amenaza de modificar información mientras viaja por la red o de modificar archivos físicos en la computadora víctima.
Repudiation (repudio)	Posibilidad de eliminar archivos y denegar el acceso a ellos, o impedir el uso de programas.
Information disclosure (revelación de información)	El hecho de mostrar demasiados datos en una página Web puede brindar información crítica al agresor. Es preferible usar comentarios tales como "Nombre de usuario o contraseña incorrecto" a desplegar uno para el fallo del nombre de usuario y otro para la contraseña. Si no, el atacante sabrá cuándo acertó uno u otro.
Denial of service (denegación de servicios)	Este ataque tiene el objetivo de saturar un servidor y anular su capacidad de respuesta. El envío de peticiones HTTP masivas o paquetes ICMP falsos logrará bloquear el servidor.
Elevation of privileges (elevación de privilegios)	Trata de explotar problemas de memoria (buffer overruns) para obtener privilegios administrativos en los equipos.

D.R.E.A.D.

Un modelo para cuantificar el impacto y la severidad de las amenazas, que nos ayuda a tener una visión completa.

Una vez identificadas las amenazas, es fácil darse cuenta de que no todas tienen probabilidades de hacerse efectivas y, al mismo tiempo, de que el efecto que podrían causar tal vez no sea lo suficientemente dañino como para tomarlas en consideración.
Al igual que S.T.R.I.D.E., D.R.E.A.D. (temor) representa una categoría por cada letra que compone la palabra, las cuales se describen en la Tabla 4.

Con el objetivo de cuantificar cada amenaza, cada una de ellas debe ser sometida al análisis propuesto por D.R.E.A.D. Una forma común es colocar por cada punto un valor numérico en rangos de 1 a 10, donde 1 representa una menor posibilidad y 10, la mayor. La suma total indicará la visión que tenemos sobre dicha amenaza, de modo que podremos priorizarlas y atacarlas individualmente.

Podemos ver en la Tabla 5 un ejemplo aplicado a un sitio de comercio electrónico.

En el caso propuesto, los factores podrían variar dependiendo del software que usemos para administrar el servidor. Como ejemplo, en "denegación de servicios", colocamos un valor bajo en la capacidad de reproducir este problema debido a que servicios como IIS (*Internet Information Services*) poseen la característica de detectar este tipo de ataques y rechazarlos.
Si bien en este momento ya sabemos cómo analizar y cuantificar las amenazas, en las siguientes páginas veremos en mayor profundidad los problemas más comunes en los que solemos incurrir cuando desarrollamos software, y aún más importante, cómo contrarrestarlos.

Tabla 4 | Siglas de D.R.E.A.D.

Damage (daño)	Se refiere a la cantidad de daño que puede causar al sistema.
Reproducibility (reproductibilidad)	Cuán fácil puede ser reproducirlo.
Exploitability (explotabilidad)	Cuán fácil es para los "hackers" aprovecharse de esta vulnerabilidad.
Affected users (usuarios afectados)	Representa la cantidad de usuarios afectados por el problema. A mayor cantidad, mayor es el problema.
Discoverability (descubrimiento)	Cuán posible es descubrir la falla.

Tabla 5 | Algunas amenazas y el análisis aplicando D.R.E.A.D.

Amenaza	D	R	E	A	D	Total
Robo de cuentas de usuarios en el servidor	8	4	2	7	0	21
Denegación de servicios	9	2	2	9	1	23

Ataques

Por cada ataque existe una forma de contrarrestarlo. En las siguientes páginas analizaremos los peligros y cómo defendernos.

Como comentamos anteriormente, muchas de las vulnerabilidades son introducidas en el momento de codificar nuestras aplicaciones. Al mismo tiempo, éstas son la puerta de entrada para un ataque. Esto quiere decir que, si bien una aplicación correctamente codificada puede seguir siendo objeto de ataques, las posibilidades de ser vulnerada serán mínimas.
Las aplicaciones Web son propensas a sufrir la mayor cantidad de ataques, por lo que debemos prestar especial atención a ellas.

Scripting

Aunque ya hicimos una breve introducción a este tipo de ataques, vale dedicarle un poco más de tiempo, en especial, por lo fácil que resulta para el atacante explotar esta vulnerabilidad.
Este ataque va mucho más allá de mostrar un simple mensaje de alerta en un foro en Internet: puede ser nocivo al punto de que la víctima nunca se entere de que ha sido afectada.
Recordemos el código propuesto anteriormente:

VB.NET:

```
Protected Sub Button1_Click(ByVal sender As
Object, ByVal e As System.EventArgs) Handles
Button1.Click
  Response.Write(Me.TextBox1.Text)
End Sub
```

C#:

```
protected void Button1_Click(object sender,
EventArgs e)
{
  Response.Write(this.TextBox1.Text);
}
```

Nuestra página Web sólo se limita a escribir en el HTML lo que el usuario introdujo en el cuadro de texto de nuestra aplicación.
El código HTML resultante es equivalente al siguiente:

```
<script>
  alert('Pagina Hackeada')
</script>
<html>
<head><title></title></head>
<body>
 <form name="form1" method="post" action=
 "Default.aspx" id="form1">
  Escriba su nombre:
  <textarea name="TextBox1" rows="3" cols=
  "20" id="TextBox1"></textarea>
  <input type="submit" name="Button1" value=
  "Saludar!" id="Button1" />
</form>
</body>
</html>
```

¿Que es phishing?

Este término deriva de la palabra en inglés "fishing", que significa "pescar". La letra "f" es reemplazada por "p", que proviene de otra palabra en inglés, "password", con la cual se forma la frase "password fishing", que traducido al español significa "pesca de contraseñas"; literalmente, la traducción acertada sería "a la pesca de contraseñas".

Como podemos ver, el código JavaScript se escribió tal como lo introdujimos. Si esta página hubiera almacenado el texto introducido por el hacker en una base de datos o de alguna otra manera física (blog, foro, etc.), y por cada usuario que visitara dicha página mostrara lo que el hacker almacenó, cada uno de ellos vería este mensaje de alerta.

Ahora extendamos este ataque. Tratemos de suplantar la identidad de un sitio Web para que el usuario crea que sigue en el original. Esto se lograría fácilmente con sólo redirigir al usuario de un sitio Web a otro falso, con un formato original. Una vez dentro del sitio falso, éste se encargaría de requerir la introducción del nombre de usuario y la contraseña del visitante para revalidar la cuenta, la cual lo habilitaría para seguir operando dentro del sitio.

Si el usuario cae en esta trampa, el hacker acaba de conseguir una cuenta de usuario para el sitio Web original. Esta técnica también es conocida como phishing.

> Las aplicaciones Web son propensas a sufrir la mayor cantidad de ataques, por lo que debemos prestar especial atención a ellas.

Si revisamos el código HTML anterior, el script de JavaScript para lograr esto es minúsculo. El agresor no necesita más que una línea de código para ejecutar su plan:

```
<script>
   location = "http://www.ServidorAgresor.com/
   PaginaFalsa.aspx";
</script>
<html>
<head><title></title></head>
...
```

Scripting

FIGURA 004 | Utilizando métodos de ataques como el scripting, un "hacker" puede obtener información confidencial de los usuarios.

> Nunca confiemos en lo que un usuario nos está enviando desde una entrada de texto.

Existen otras posibilidades de agresión, como la de objetos embebidos en páginas Web. En la Tabla 6 podemos ver las más recurrentes.

Es común combinar código JavaScript con estos objetos. La idea es que, mediante JavaScript, se manipule el objeto para que reaccione de un modo u otro. Por ejemplo, configurar el servicio Web desde donde se obtendrá o enviará la información. Por lo tanto, el agresor, usando las técnicas antes vistas, podría crear su propio código JavaScript que manipule el objeto, modificando su comportamiento. Un usuario desprevenido podría estar enviando información directamente al agresor sin percatarse de esta situación.

La solución para el scripting

Afortunadamente, contamos con una solución, que aparece por descarte: "nunca confiemos en lo que un usuario nos está enviando desde una entrada de texto". Esta afirmación puede parecer extremista, pero sin temor a equivocarnos, podemos aseverar que es aplicable no sólo a este caso, sino también a otros que veremos más adelante.

Esta desconfianza radica en que debemos evaluar cada entrada del usuario y, así, detectar cualquier posible intento de amenaza.
Una forma de lograrlo es usar el método **HtmlEncode**, que se encuentra dentro del objeto **Server** de ASP.**NET.**
Veamos el código modificado para este caso:

VB.NET:

```
Protected Sub Button1_Click(ByVal sender As
Object, ByVal e As System.EventArgs) Handles
Button1.Click
  Response.Write(Server.HtmlEncode(Me.
  TextBox1.Text))
End Sub
```

C#:

```
protected void Button1_Click(object sender,
EventArgs e)
{
  Response.Write(Server.HtmlEncode(this.
  TextBox1.Text));
}
```

HtmlEncode remplazará todos aquellos caracteres especiales por sus equivalentes

Tabla 6 | Ataques usando objetos embebidos

Applets de Java	Usadas en animaciones, acceso a la máquina cliente, entre otros.
Películas de Flash	Generalmente usadas para brindar contenido animado en páginas Web, aunque en versiones avanzadas estos componentes permiten interactuar con servicios Web.
Objetos ActiveX	Ésta es la propuesta por parte de Microsoft, que simula el comportamiento de los applets de Java. La particularidad de los ActiveX es la versatilidad con la que pueden usar recursos del cliente. Claro que para lograrlo, se requiere la aprobación del usuario.

HTML. Como resultado, el código JavaScript será evaluado como simples etiquetas HTML y no como código para ser ejecutado.
En todo caso, ASP.NET posee la capacidad de preanalizar todas las entradas de texto en nuestras páginas y detectar cuáles serían potencialmente peligrosas. Para lograrlo, debemos modificar el atributo **@Page** de nuestra página de la siguiente forma:

```
<%@ Page Language="VB" AutoEventWireup
="false" CodeFile="Default.aspx.vb"
Inherits="_Default" ValidateRequest="true" %>
```

Una ventaja de ASP.NET es que no necesitamos definir por cada página de nuestra aplicación el atributo **ValidateRequest**, ya que éste, sin estar escrito, por defecto está asignado con el valor en True.

SQL Injection

El SQL Injection, o Inyección SQL, representa otro gran problema, pero en este caso, con consecuencias directas para nuestra aplicación Web.
Con este ataque, el agresor pretende, a través de comandos SQL o lenguaje de consultas, obtener información de nuestra base de datos, ingresar en zonas restringidas o, simplemente, destruir por completo nuestra base de datos.

Para entender la inyección de SQL, veamos un ejemplo típico de conexión a una base de datos, y la ejecución de una consulta SQL:

VB.NET:

```
Dim Conexion As New System.Data.SqlClient.
SqlConnection("...")
Dim Comando As New System.Data.SqlClient.
SqlCommand()
```

> Con SQL Injection, el agresor pretende, a través de comandos SQL o lenguaje de consultas, obtener información de nuestra base de datos.

```
Dim SQL As String

Comando.Connection = Conexion
Comando.CommandType = Data.CommandType.Text

SQL = "SELECT * FROM Clientes WHERE Nombre='"
& Me.TextBox1.Text & "'"

Comando.CommandText = SQL
...
```

C#:

```
System.Data.SqlClient.SqlConnection Conexion
= new System.Data.SqlClient.SqlConnection
("...");
System.Data.SqlClient.SqlCommand Comando =
new System.Data.SqlClient.SqlCommand();
string SQL;

Comando.Connection = Conexion;
Comando.CommandType = Data.CommandType.Text;

SQL = "SELECT * FROM Clientes WHERE Nombre='"
+ this.TextBox1.Text + "'";

Comando.CommandText = SQL;
...
```


Una vez más, el agresor hará uso de las entradas de texto para tratar de acoplar su código

La denegación de servicios es una agresión dirigida a inutilizar la capacidad de un sitio Web o servidor.

SQL al de nuestra consulta, y dejar que nuestro código lo ejecute.

Supongamos que la entrada propuesta por el atacante sea similar a ésta:

```
' OR 1=1 –
```

Como nosotros estamos concatenado el texto del cuadro directamente sobre nuestra consulta, el contenido de la variable SQL quedaría como sigue:

```
SELECT * FROM Clientes WHERE Nombre=''
or 1=1 –'
```

El objeto **Command** ejecutará la consulta sobre el motor de base de datos con este acople, que tendrá como resultado la obtención de todos los registros de la tabla. Si bien la entrada anterior no representa un riesgo alto, vamos a aplicarla al módulo de autentificación de usuarios de nuestra aplicación. En este caso, no importa si el agresor conoce un nombre de usuario y una contraseña, ya que la igualdad final asegura la obtención de uno o más usuarios válidos de la aplicación, ingresando en ella y suplantando la identidad del primero en la lista. Como consecuencia, el agresor podría usar la misma consulta para ejecutar el comando **Drop Table** o **Drop Database**, y causar daños irreparables. Una vez más, la solución radica en la desconfianza de los datos brindados por el usuario, y el remplazo de caracteres no válidos antes de llevarlos a la consulta. Otra solución, más sofisticada, se nutre del uso de parámetros SQL en vez de la concatenación de datos a consultas planas.

ⓘ Blaster: un virus que usaba la denegación de servicios

A fines del año 2002, se popularizó un virus informático llamado iWorm.Blaster, un gusano que tenía la particularidad de autopropagarse de una manera increíble, aprovechando una vulnerabilidad del sistema operativo Windows. Él mismo realizaba un escaneo de las IPs más cercanas a la de la máquina infectada y se autotransmitía para infectar a las victimas. Pero el verdadero problema era que este gusano comenzaba a enviar paquetes erróneos al sistema operativo, lo que generaba una sobrecarga del buffer que obligaba al usuario a reiniciar la computadora.

Denegación de servicios

La denegación de servicios es una agresión dirigida a inutilizar la capacidad de un sitio Web o servidor, de tal forma que éste quede, justamente, inútil o inhabilitado para trabajar. Este ataque se lleva a cabo mediante la petición de respuestas por parte del servidor en cantidades que éste no puede manejar. Tengamos en cuenta que por cada usuario que visita una página Web en un servidor, éste de-

be realizar determinadas tareas para poder responder a esa petición con la página en cuestión, como:

- Consumo de memoria
- Ruteo de paquetes
- Consumo de ancho de banda
- Uso de disco duro para lectura de archivo

Entonces, por cada petición, el servidor necesitará un tiempo determinado para poder responder. Pero si existen cientos de miles de peticiones simultáneas, éste llegará a su límite rápidamente, lo que le impedirá responder a más peticiones hasta no liberar las anteriormente requeridas.
Es común, además, que en este tipo de ataques se incluyan el envío de paquetes IP mal formados. En ellos se acostumbra, dentro de la trama IP, a modificar el remitente de la solicitud, de modo que cuando un servidor trata de responder a este remitente, pierde mucho más tiempo porque éste, en realidad, no existe. El servidor, en este caso, continuará intentando enviar la respuesta e, indudablemente, consumirá más de los recursos nombrados.

En este momento ya contamos con conocimientos sólidos sobre las amenazas de las que pueden ser víctimas nuestras aplicaciones Web, y, más aún, sabemos cómo contrarrestarlas. En los próximos capítulos trataremos el problema a nivel de hardware y arquitectura, los que nos terminará otorgando una visión integral de cómo protegernos antes agresores externos.

Denegación de servicio

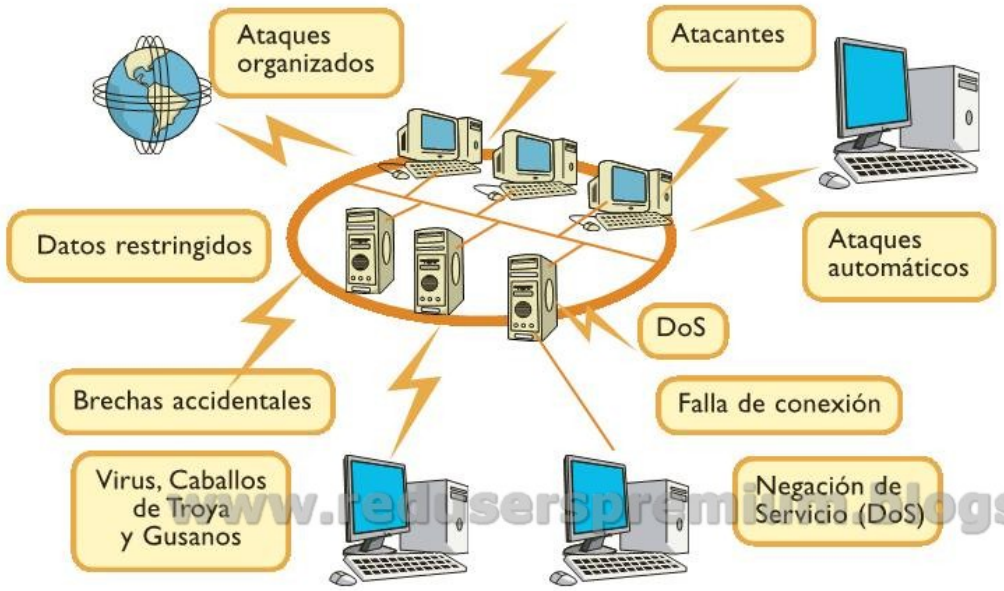

FIGURA 005 | Observamos aquí las diferentes formas en las que se puede lograr una denegación del servicio: ataques del tipo organizado, automático, etc.

Plataformas de protección

Al momento de proteger nuestro trabajo, también deberemos brindar seguridad a la plataforma de red que utilicemos.

Al principio de este capítulo dedicado a seguridad, hemos comprobado que no todos los ataques sufridos tienen como medio las aplicaciones que desarrollamos, sino que muchos están enfocados en la penetración de redes corporativas o en la anulación de la capacidad de brindar los servicios que, por ejemplo, un sitio Web puede proporcionar, esto último, a través de la denegación de servicios.

> No todos los ataques sufridos tienen como medio las aplicaciones que desarrollamos, sino que muchos están enfocados en la penetración de redes corporativas.

Estos ataques hacen uso de los protocolos y puertos empleados para brindar el servicio en cuestión. En el caso de puertos, éstos deben ser considerados como puertas abiertas en los equipos, a la espera de información. Estos puertos son las vías de comunicación que enlazan el equipo servidor (*host*) con el cliente (*client*). Muchos de ellos sirven, por ejemplo, para ejecutar comandos en el servidor o su administración remota, hasta permitir la conversación por medio de un programa de chat o enviar un correo. Pero así como proporcionan un servicio específico, una administración incorrecta o el desconocimiento de su existencia pueden otorgarle a un agresor una puerta de entrada a nuestro sistema o red empresarial. En la Tabla 7 podemos ver una lista de los puertos más comunes, y sus direcciones

Tabla 7 | Lista de los puertos usados con más frecuencia

Puerto	Uso común
20	FTP data
21	FTP (File Transfer Protocol)
23	Telnet
25	SMTP (envío de correos electrónicos)
53	DNS (Domain Name Service)
68	DHCP (Dynamic Host Control Protocol)
80	http
110	POP3 (recepción de correos electrónicos)
194	IRC (Internet Relay Chat)
443	SSL (Secure Socket Layer)
1433	Microsoft SQL Server
1521	Oracle SQL
31337	**BackOrifice (troyano)**

más conocidas. Se trata de las direcciones (numeración) más comunes porque los servicios que se brindan por medio de ellos podrían ser otorgados por algún otro. Tomemos un ejemplo hipotético de un sitio Web: **www.MiSitio.com:7070**. En este caso, la asignación de los dos puntos y el número 7070 le dicen al navegador Web que lo utilice como puerto de entrada para la página que requerimos, en vez del convencional puerto 80. Finalmente, la tabla también hace referencia a un último puerto, que era usado por un conocido troyano de principios del año 2000. Este programa malicioso se instalaba en los equipos víctima abriendo un puerto de escucha, con el objetivo de que el atacante pudiera conectarse a la máquina y manipularla en forma remota.

Por todo lo visto, podemos entender que existen mecanismos para poder cerrar puertos de escucha, así como para auditarlos.

Comúnmente, se cree que un firewall es un equipo de hardware, cuando en la actualidad existen programas (software) que cumplen la misma función.

Firewalls

Los firewalls son uno de estos mecanismos a los que nos referíamos anteriormente. Actúan como barrera o filtro entre nuestra red y las peticiones externas.

Por un lado, se encargan de auditar los puertos y protocolos expuestos antes de que lleguen a nuestra red. Por ejemplo, un firewall podría tener la tarea de escuchar todas las peticiones

Protección vía firewall

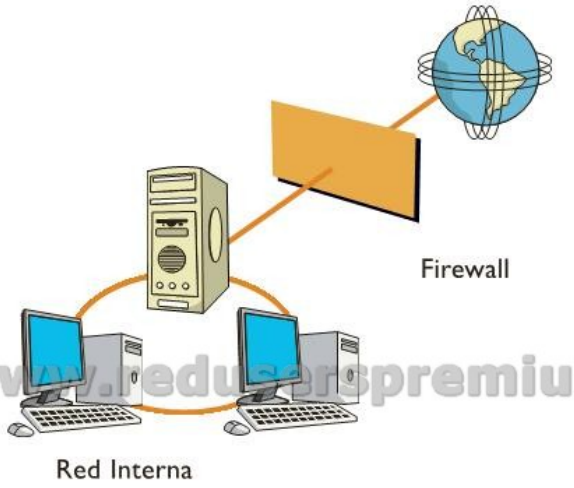

FIGURA 006 | Un firewall se interpone entre una red externa, como Internet, y nuestra red corporativa.

UN FIREWALL NO ES UN MECANISMO POR EL CUAL IMPEDIREMOS EL NORMAL DESEMPEÑO DE LA RED, SINO QUE ES UNA MANERA DE CONTROLAR QUÉ PASA POR ELLA.

realizadas sobre el puerto 80 (HTTP), por navegadores de Internet sobre nuestro sitio Web. Una vez que esta petición es validada, podría redirigirla a nuestro servidor, oculto detrás del firewall, pero en otro puerto, el cual por cuestiones de seguridad espera peticiones HTTP por el 7070. Este análisis previo y redirección de la petición nos dan cierto grado de seguridad, al no dejar expuesto el servidor en sí, sino que primero un agresor debería atravesar esta barrera inicial, lo que agrega mayor dificultad para un ataque.
De cualquier manera, un firewall no tiene como único objetivo analizar y redirigir peticiones, sino más bien reducir las superficies de ataques posibles. De este modo, como dijimos al principio, logra anular o cerrar puertos y protocolos innecesarios, que quizá por error u omisión hemos dejado abiertos en los equipos que conectamos directamente a la red o a Internet.
Comúnmente, se entiende que un firewall es un hardware intermediario entre la conexión a Internet y nuestra red. Si bien existen equipos físicos preparados para realizar estos trabajos, en la actualidad los firewalls también pueden ser software, programas en diferentes escalas para instalar en distintos sistemas operativos. Un firewall por software de gran escala es ISA Server, de Microsoft. Es ideal para cubrir una red corporativa, tanto para las peticiones de entrada como para la auditoría de salidas de la misma red. De cualquier manera, también existen firewalls personales, de menor escala y capacidad, pero igualmente eficientes, como el firewall de Microsoft incluido en el Service Pack 2 de Windows XP, y el firewall incorporado en Windows Vista.
En resumen, un firewall no es un mecanismo por el cual impediremos el normal desempeño de la red y la forma en que los equipos se comunican, sino que es una manera de controlar qué pasa por la red, tanto por parte de las peticiones entrantes –aquellas que se originan en Internet e impactan sobre nuestros equipos–, como de aquellas que se originan dentro de la red y viajen fuera del dominio.

DMZ

Otro esquema que encontramos dentro de las plataformas de protección, a nivel de arquitectura, son las zonas desmilitarizadas, o DMZ, por su sigla en inglés (***De**Militarized **Z**one*).
Este esquema plantea el concepto de seccionar la red en partes, con rangos de IPs distintos para cada una de las áreas sensibles de nuestra organización. Esto significa separar la red corporativa de aquella que brinda servicios sobre Internet.
En un esquema simple, un dispositivo de firewall podría conmutar las peticiones entre dos redes; de este modo, si se realiza una petición al servidor de aplicaciones Web desde Internet, el firewall la redirigirá a la zona o segmento de red donde se encuentra este servidor, y así impedirá que la petición traspase hacia la red corporativa.

Argentina $8,90 (recargo al interior $0,20) / México: $45

USERS

Microsoft

Curso teórico y práctico de programación

Desarrollador .net

Con toda la potencia de **Visual Basic .NET** y **C#**

La mejor forma de aprender a programar desde cero

★★★★★

Basado en el programa Desarrollador Cinco Estrellas de Microsoft

17

SALIDA LABORAL
La mejor preparación para rendir los exámenes
GARANTIZADA

Seguridad

Plataforma de protección para componentes, bases de datos y servicios

Framework 3.0

Todas las novedades - WPF, WCF y WF

ISBN 978-987-1347-43-8

9 789871 347438 00017