Sesión 3 : Prácticas de codificación segura

Introducción a la seguridad de aplicaciones y mejores prácticas en codificación #

Mauricio Sotelo

- Ingeniero de Seguridad Nivel III
- Seguidor de las buenas prácticas, marcos de trabajo y estándares de seguridad.
- Amo las actividades al aire libre y hacer todo con música de fondo.
- https://calendly.com/mauricio_sotelo

Recomendaciones importantes

Identifícate en Zoom usando tu nombre y apellido.

Silencia tu micrófono durante el curso.

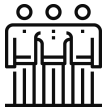
Utiliza el chat para hacer preguntas en la sección asignada para ello.

Centra tus preguntas en el tema presentado.

Mantén tu cámara encendida durante toda la sesión.

Academy Codigo de Conducta

Sé respetuoso, no hay preguntas o ideas malas.



Sé empático y paciente.

Sé cuidadoso con las palabras que eliges.

Objetivo

Al final de esta sesión podrás:

- Entender y poner en práctica los conceptos revisados de seguridad en la programación.

Tabla de Contenidos

Seguridad en la Programación

Antecedentes

Breve Historia de la Ciberseguridad:

- **1970s:** Primeros virus informáticos (p.ej. Creeper).
- **1980s:** Auge del malware, inicio de los antivirus.
- **2000s:** Cibercriminalidad y ataques masivos (p.ej. ILOVEYOU, Slammer).
- **2010s:** Guerra cibernética y espionaje (p.ej. Stuxnet, Ataque a Sony).
- **2020s:** Aumento de ataques a cadenas de suministro Maersk, incidentes de SolarWinds y ataques de ransomware a infraestructuras críticas.

2030: Qué podríamos esperar?

Problemas Comunes:

- **Inyección de código.**
- **Vulnerabilidades de día cero.**
- **Ataques de fuerza bruta.**

Principios Generales de Programación Segura

- Seguridad desde el diseño (Security by Design).
- Principio de menor privilegio.
- Defensa en profundidad.
- Segregación de funciones y separación de ambientes (desarrollo, prueba, producción).

Seguridad en la Programación

¿Qué es?

- Consiste en escribir el código pensando en posibles ataques.
- Conocer los ataques para poder prevenirlos en código.
- Considerar la integridad, la confidencialidad de los datos en todo momento.
- Programar considerando la disponibilidad y autenticidad de todos los sistemas y servicios que usemos.

¿Para qué sirve?

- Proteger la información.
- Mantener la confianza del cliente y la reputación corporativa.
- Cumplimiento legal.
- Previene daños y pérdidas financieras.
- Responsabilidad Ética.

Validación de las entradas (Input Validation)

Es el proceso mediante el cual se verifica y valida cualquier dato ingresado a una aplicación antes de procesarlo.

Se lleva a cabo para asegurarse de que los datos son correctos, significativos y seguros antes de ser utilizados.

Es crucial por varias razones:

- **Seguridad**
- **Integridad de los datos**
- **Experiencia del usuario**

Técnicas para una validación efectiva

- Usar listas de permitidos (allow-lists) en lugar de listas de bloqueo (block-lists).
- Validar Tipo, Formato, Longitud y Rango.
- Evitar la inyección de código mediante la validación de entrada.

Algunas Herramientas

- Microsoft Data Annotations
- FluentValidation
- NUnit
- XUnit

Codificación de Salida (Output Encoding)

Importancia de tratar todos los datos como no confiables:

Todo dato, ya sea originado por el usuario, proveniente de una base de datos o incluso de servicios de terceros, debe ser tratado como potencialmente malicioso o no confiable. Las razones incluyen:

- **Amplia superficie de ataque**
- **Ataques sofisticados**
- **Principio de defensa en profundidad**

Codificación de Salida

- HTML: <, >, y & → <, >, y &
- URLs: espacios → %20
- Bases de Datos

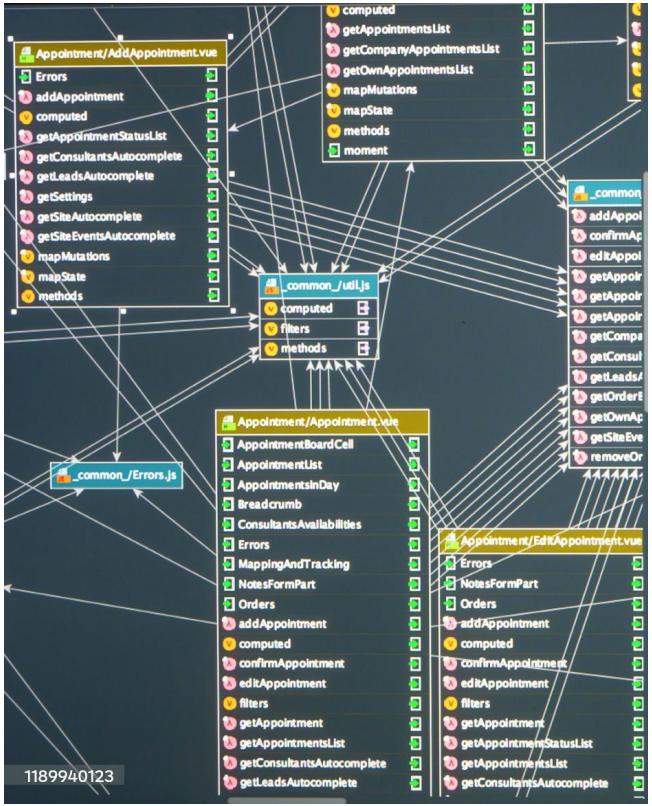
¿Para qué sirve?

- Evitar la inyección de etiquetas o scripts maliciosos
- Para ser transmitidos de manera segura dentro de una URL
- Evitar inyección SQL
- Evitar ataques de Cross-Site Scripting (XSS)

Cross Site Scripting (XSS)

Inyección SQL

- Es una técnica en la que un atacante puede ejecutar comandos SQL maliciosos en una base de datos al **"inyectar"** estos **comandos a través de la entrada** proporcionada por el usuario en una aplicación.
- Esta vulnerabilidad se produce cuando la aplicación no valida o sanitiza adecuadamente las entradas del usuario antes de usarlas en una consulta SQL.



Inyección SQL

El código inyectado:

```
User: admin
Password: abc' OR 1=1;--
```

- **abc'** cierra la cadena de texto original.
- **OR 1=1** es una condición siempre verdadera, lo que significa que se recuperarán todos los registros de la tabla.
- ; marca el fin de una instrucción SQL.
- **--** comenta el resto de la consulta original, si es que hay algo después.

El código SQL resultante sería:

```
SELECT * FROM users WHERE username = 'admin' AND password = 'abc' OR 1=1;--';
```

Ataques de Fuerza Bruta

- Es una técnica en la que un atacante intenta adivinar contraseñas, claves o información de autenticación mediante intentos repetidos.
- Esta vulnerabilidad se produce cuando la aplicación no implementa medidas adecuadas para prevenir o limitar el número de intentos que un usuario puede realizar en un corto período de tiempo.

Descanso

5 minutos.

Prácticas Avanzadas

Uso de encabezados de seguridad HTTP para proteger las aplicaciones web

Uso de encabezados de seguridad HTTP para proteger las aplicaciones web:

- Content-Security-Policy (CSP).
- Strict-Transport-Security (HSTS).
- X-Content-Type-Options.
- X-Frame-Options.

Implementar estos y otros encabezados de seguridad reduce significativamente la superficie de ataque de una aplicación web.

Gestión segura de sesiones y autenticación

Manejar correctamente las sesiones y la autenticación es fundamental para mantener la seguridad de las aplicaciones. Algunas prácticas recomendadas incluyen:

- Utilizar identificadores de sesión aleatorios y difíciles de adivinar.
- Regenerar estos identificadores después de un cierto tiempo o después de que un usuario inicie o cierre sesión.
- Limitar la duración de las sesiones y usar tiempo de inactividad para reducir el de exposición.
- Evitar almacenar información sensible directamente en las cookies de sesión.

```
using System;
using System.Web;
using System.Web.SessionState;

public class SessionManagement
{
    // Duración máxima de la sesión (por ejemplo, 30 minutos)
    private const int SessionMaxDuration = 30;  // en minutos

    // Tiempo máximo de inactividad antes de regenerar el identificador de sesión
(por ejemplo, 10 minutos)
    private const int SessionRegenerationTime = 10;

    // Hora en que se creó o regeneró por última vez el identificador de sesión
    private DateTime LastSessionRegeneration;

    public SessionManagement()
    {
        // Al iniciar una nueva sesión, generamos un identificador aleatorio y
guardamos la hora
        HttpContext.Current.Session.SessionID = GenerateRandomSessionId();
        LastSessionRegeneration = DateTime.Now;
    }
```

```
  private string GenerateRandomSessionId()
    {
        // Generar un identificador de sesión aleatorio y difícil de adivinar
        return Guid.NewGuid().ToString();
    }
    public void CheckSession()
    {
        // Verificar si ha pasado el tiempo de regeneración del identificador de
sesión
        if (DateTime.Now.Subtract(LastSessionRegeneration).Minutes >
SessionRegenerationTime)
        {
            HttpContext.Current.Session.SessionID = GenerateRandomSessionId();
            LastSessionRegeneration = DateTime.Now;
        }
        // Verificar si ha pasado el tiempo máximo de duración de la sesión
        if
(DateTime.Now.Subtract(HttpContext.Current.Session["SessionStartTime"]).Minutes >
SessionMaxDuration)
        {
            HttpContext.Current.Session.Abandon();  // Finalizar la sesión
        }
    }
```

```
    public void StoreSensitiveDataInSession(string key, object value)
    {
        // Evitar almacenar información sensible directamente en cookies. En su
lugar, guardamos en la sesión.
        HttpContext.Current.Session[key] = value;
    }

    public object RetrieveSensitiveDataFromSession(string key)
    {
        // Recuperar información de la sesión
        return HttpContext.Current.Session[key];
    }
}
```

Encripción o Cifrado

Almacenamiento seguro de contraseñas: hashing y salting:

Almacenar contraseñas en texto plano es una vulnerabilidad crítica. Para almacenarlas de forma segura, es esencial:

- **Hashing**
- **Salting**

Veremos un ejemplo de cómo hacer hashing de contraseñas en C# utilizando la biblioteca BCrypt.Net, que implementa el algoritmo bcrypt.

Debes instalar la biblioteca. Puedes hacerlo usando NuGet:

```
Install-Package BCrypt.Net-Next
```

```
using BCrypt.Net;

public class PasswordManager
{
    public static string HashPassword(string password)
    {
        // Hash y sal de la contraseña usando bcrypt
        return BCrypt.HashPassword(password);
    }

    public static bool VerifyPassword(string enteredPassword, string storedHash)
    {
        // Verificar si la contraseña ingresada coincide con el hash almacenado
        return BCrypt.Verify(enteredPassword, storedHash);
    }
}
```

```
// Uso del código
public class Program
{
    public static void Main()
    {
        string password = "SuperSecret123!";

        // Crear el hash de la contraseña
        string hashedPassword = PasswordManager.HashPassword(password);
        Console.WriteLine($"Hashed Password: {hashedPassword}");

        // Verificar la contraseña
        bool isVerified = PasswordManager.VerifyPassword("SuperSecret123!",
hashedPassword);
        Console.WriteLine($"Password Verification: {isVerified}");
    }
}
```

Uso de HTTPS y TLS para transmisión segura de datos

Algunas recomendaciones incluyen:

- Utilizar certificados válidos y mantenerlos actualizados.
- Configurar el servidor para usar versiones seguras y actualizadas de TLS.
- Evitar protocolos inseguros como SSL

Cifrado de la información

En tránsito:

- Usa HTTPS para asegurar la integridad y confidencialidad de los datos..
- Si HTTPS no es viable, aplica cifrado con AES-256.

En reposo:

- Cifra la información crítica o sensible.
- Para datos como nombres de usuario, usa cifrado AES-256.
- Para contraseñas, opta por hash seguro como bcrypt o Argon2 en lugar de SHA-256.

En respaldo:

- Cifra los respaldos para prevenir filtraciones.
- Guarda las llaves de cifrado separadamente del respaldo.

OWASP en acción

Principios de OWASP

- Autenticación segura y autorización basada en roles
- Consultas parametrizadas o preparadas
- Manejo de sesiones y tokens
- Validación y sanitización
- Encabezados de seguridad
- Escaneos y pruebas de seguridad
- Actualización de componentes
- Monitoreo y registro

¿Preguntas?

Gracias.