Sesión 13 - Mitigación y remediación de Vulnerabilidades

Guillermo Esguerra

@esguerrag - GitHub

Acerca de mí

Guillermo Esguerra

SRE IV - Tech Lead

- Amante de Linux trabajando desde un Mac.
- Creado y criado en la industria bancaria y fintech.
- Experto en infraestructura y observabilidad.
- Me gusta dañar algunas cosas para volverlas a reparar.

Puntos importantes

Identifícate en Zoom utilizando tu nombre y apellido.

Mantén tu micrófono apagado durante el transcurso de la sesión.

Utiliza el chat para hacer tus preguntas durante la sección de Q&A.

Procura enfocar tus preguntas al tema presentado.

Apaga tu cámara en caso de tener problemas con tu conexión.

Código de conducta

Sé respetuoso, no hay malas preguntas o ideas.

Sé cordial y paciente.

Sé cuidadoso con tus palabras.

Objetivo

Al final de esta sesión podrás:

- Reconocer los posibles puntos de quiebre en nuestras soluciones.
- Minimizar el área de impacto de posibles fallos.
- Implementar buenas prácticas de desarrollo seguro.

Random.org Time

Reglas del juego:

- En el momento de cualquier pregunta, un nombre sera seleccionado en random.org
- Si no sabes, salta a la siguiente persona
- Se puede escoger entre la cabeza o la cola de la lista.

Tabla de Contenidos

Principios de mitigación

Estrategias de remediación

Integración con el ciclo de desarrollo

Repaso rápido

Tipos de vulnerabilidades

- Vulnerabilidades por diseño
- Vulnerabilidades por Implementación
- Vulnerabilidades por configuración

Estrategias de análisis de código

- Análisis Estático de Código Fuente (SAST)
- Análisis Dinámico de Aplicaciones (DAST)
- Revisión Manual del Código

Principios de mitigación

Principios de mitigación

- Principio de Menor Privilegio
- Defensa en Profundidad
- Segregación de Ambientes
- Codificación Segura

Principios de mitigación

- Validación de Entrada y Saneamiento de Datos
- Actualización y Parcheo Constante
- Auditoría y Monitoreo
- Respuesta a Incidentes

Principio de Menor Privilegio

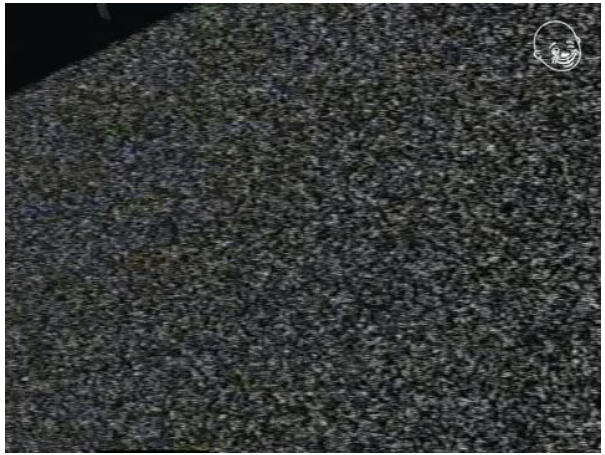
El conjunto más limitado de privilegios necesario para completar su función.

Defensa en Profundidad

Múltiples capas de seguridad para que si una capa es comprometida, las restantes sigan proporcionando protección.

Segregacion de ambientes

Mantener ambientes de desarrollo, prueba y producción separados para evitar que las vulnerabilidades se propaguen del desarrollo a la producción.



Asegurar por Diseño

Incorporar la seguridad desde las primeras etapas del diseño de software, garantizando la seguridad.

Codificación segura

Seguir guías y normativas de codificación segura para prevenir vulnerabilidades comunes.

Actualización y Parcheo Constante

Mantener el herramientas y librerías de software actualizado con los últimos parches de seguridad para proteger contra vulnerabilidades conocidas.

Estrategias de remediación

Estrategias de remediación

- Refactorización de Código
- Revisión de Código
- Implementación de Controles de Seguridad
- Gestión de Configuraciones de Seguridad
- Pruebas de Seguridad Regular
- Automatización de la Seguridad

Validación de la entrada

```
public void SaveProduct(string productName)
{
    var query = "INSERT INTO products (name) VALUES ('" + productName + "')";
    // ... ejecución de la consulta ...
}
```

Validación de la entrada

```
public void SaveProduct(string productName)
{
    if (IsValidProductName(productName))
    {
        var query = "INSERT INTO products (name) VALUES (@productName)";
        SqlCommand command = new SqlCommand(query);
        command.Parameters.AddWithValue("@productName", productName);
        // ... ejecución de la consulta ...
    }
    else
    {
        // Manejo del error o excepción
    }
}

private bool IsValidProductName(string productName)
{
    // Reglas de validación aquí
    return !string.IsNullOrEmpty(productName) && productName.Length < 50;
}
```

Implementación de controles

```
public void RegisterUser(string username, string password)
{
    var query = "INSERT INTO users (username, password) VALUES ('" + username + "', '" + password + "')";
    // ... ejecución de la consulta ...
}
```

Implementación de controles

```
public void RegisterUser(string username, string password)
{
    var hashedPassword = HashPassword(password);
    var query = "INSERT INTO users (username, password) VALUES (@username, @hashedPassword)";
    SqlCommand command = new SqlCommand(query);
    command.Parameters.AddWithValue("@username", username);
    command.Parameters.AddWithValue("@hashedPassword", hashedPassword);
    // ... ejecución de la consulta ...
}

private string HashPassword(string password)
{
    // Utilizar un algoritmo de hash adecuado con sal
    using (var sha256 = SHA256.Create())
    {
        var saltedPassword = "random_salt" + password;
        var saltedPasswordBytes = Encoding.UTF8.GetBytes(saltedPassword);
        var hashBytes = sha256.ComputeHash(saltedPasswordBytes);
        var hashString = Convert.ToBase64String(hashBytes);
        return hashString;
    }
}
```

Automatización de la Seguridad

```
using NUnit.Framework;
using System;

namespace YourApp.Tests
{
    [TestFixture]
    public class AuthenticationTests
    {
        // Simulando un servicio de autenticación
        public class AuthenticationService
        {
            public bool Authenticate(string username, string password)
            {
                // Aquí iría la lógica de autenticación real
                return username == "usuario" && password == "contraseñaSegura123";
            }
        }

        [Test]
        public void Authenticate_ShouldReturnFalse_WhenPasswordIsIncorrect()
        {
            // Arrange
            var authService = new AuthenticationService()
            var username = "usuario";
            var incorrectPassword = "contraseñaIncorrecta";

            // Act
            var result = authService.Authenticate(username, incorrectPassword);

            // Assert
            Assert.IsFalse(result, "La autenticación debería fallar con una contraseña incorrecta.");
        }
    }
}
```

¿Preguntas?

Gracias.