Sesión 12 - Identificación 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
- Explorar los tipos de alcances para reconocer las vulnerabilidades
- Definir posibles soluciones para estas

Tabla de Contenidos

Tipos de vulnerabilidades

Tipos de vulnerabilidades

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

Vulnerabilidades por diseño

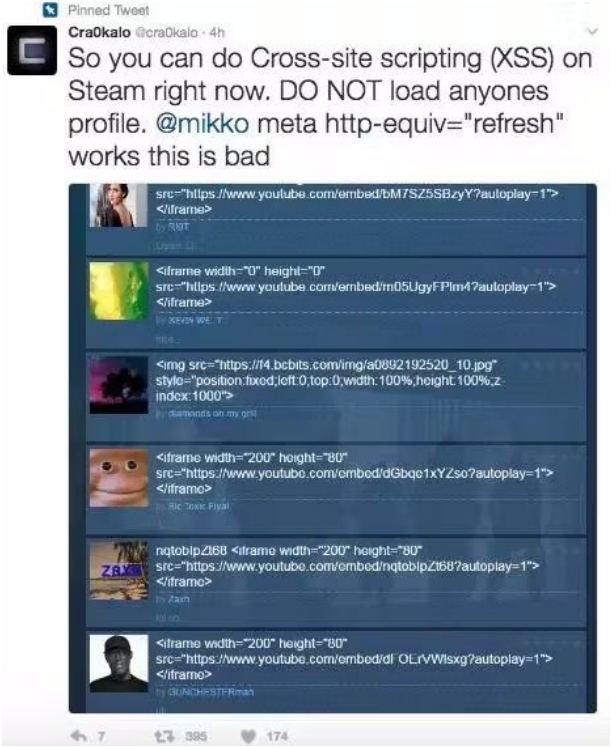
Un diseño inseguro es una falla fundamental en la concepción de la seguridad que no puede compensarse con una implementación técnica correcta.

Vulnerabilidades por diseño

Vulnerabilidades por implementación

La aplicación puede estar en riesgo si se desconocen las versiones de todos los componentes usados, si hay software vulnerable sin actualizar, si no se realiza un análisis regular de vulnerabilidades, si no se actualizan los sistemas a tiempo, o si no se prueba la compatibilidad de las bibliotecas actualizadas.

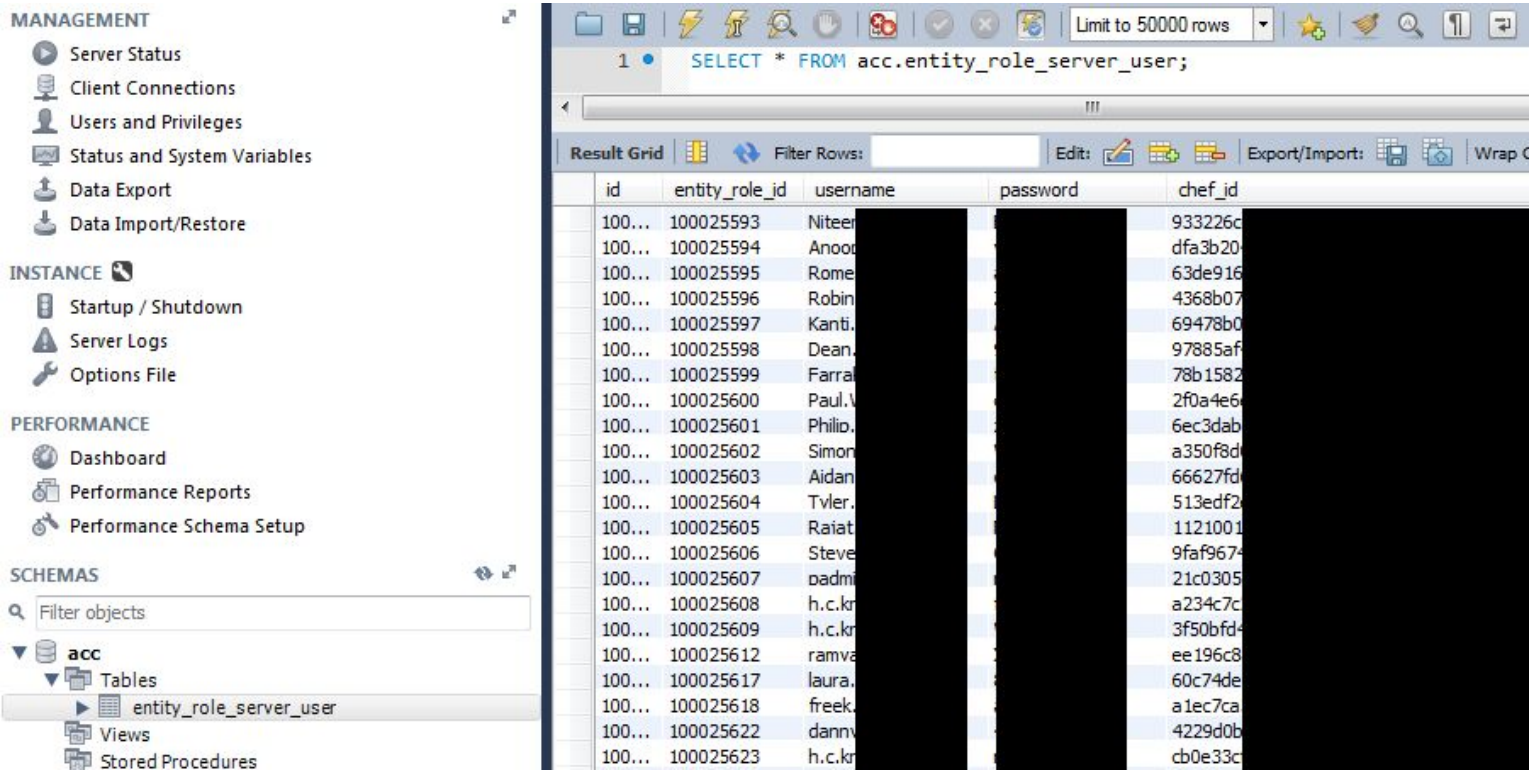
Vulnerabilidades por implementación



Vulnerabilidades por configuración

La aplicación es vulnerable debido a configuraciones de seguridad inadecuadas, funciones superfluas, cuentas predeterminadas inseguras, manejo de errores revelador, desactivación de medidas de seguridad actualizadas y uso de software desactualizado o vulnerable.

Vulnerabilidades por configuración



Estrategias de análisis de código

Estrategias de análisis de código

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

Análisis Estático de Código Fuente (SAST)

El Análisis Estático de Código (SAST, por sus siglas en inglés) es un proceso que examina el código fuente de una aplicación para identificar vulnerabilidades de seguridad, bugs y deficiencias de calidad sin ejecutar el programa.

Sonarqube

- Errores de Compilación
- Code Smells
- Bugs
- Vulnerabilidades de Seguridad
- Duplicación de Código

Sonarqube

- Complejidad Ciclomática
- Cobertura de Código
- Reglas de Estilo y Formateo
- Problemas de Seguridad y Cumplimiento

Análisis Estático de Código Fuente (SAST)

```
public class Calculator
{
    public int Add(int a, int b)
    {
        return a + b;
    }

    public int Divide(int a, int b)
    {
        return a / b; // Potencial punto de falla si b es 0
    }
}
```

Sonarqube

- **Divide()** es un bug potencial
- Validación de entrada con **b** con excepciones
- Formateo y legibilidad del código

Análisis Estático de Código Fuente (SAST)

```
public class Calculator
{
    public int Divide(int a, int b)
    {
        if (b == 0)
        {
            throw new ArgumentException("El divisor no puede ser cero.", nameof(b));
        }
        return a / b;
    }
}
```

Análisis Dinámico de Aplicaciones (DAST)

El Análisis Dinámico de Aplicaciones (DAST) es un proceso de pruebas que detecta condiciones de seguridad y vulnerabilidades en aplicaciones web en ejecución, simulando ataques externos.

Análisis Dinámico de Aplicaciones (DAST)

```
// Controlador ASP.NET que recibe un comentario de un formulario web y lo muestra en una página
[HttpPost]
public ActionResult SubmitComment(string userComment)
{
    // Guarda el comentario en la base de datos
    Database.SaveComment(userComment);

    // Redirige al usuario a la página de comentarios donde se muestran todos los comentarios, incluido el nuevo
    return RedirectToAction("Comments");
}

public ActionResult Comments()
{
    var comments = Database.GetComments();
    return View(comments);
}
```

Análisis Dinámico de Aplicaciones (DAST)

- La herramienta DAST enviaría datos de entrada al formulario de comentarios a través de solicitudes HTTP POST que incluyen scripts de JavaScript maliciosos.
- Luego, la herramienta accedería a la página de comentarios para ver si el script se ejecuta.
- Si el script se ejecuta, esto indicaría que la aplicación no está codificando adecuadamente las entradas del usuario para su visualización en HTML, lo que resulta en una vulnerabilidad XSS.

Análisis Dinámico de Aplicaciones (DAST)

```
public ActionResult Comments()
{
    var comments = Database.GetComments().Select(HttpUtility.HtmlEncode);
    return View(comments);
}
```

Estrategia de penetración y auditorías

Pruebas de Penetración

- Pen testers simulan ataques contra sistemas para encontrar vulnerabilidades que un atacante podría explotar.
- Puede incluir la inyección SQL, scripts entre sitios (XSS), y otros vectores de ataque.

Scanning de Vulnerabilidades

- Herramientas como Nessus, OpenVAS, y Qualys escanean los sistemas para detectar vulnerabilidades conocidas.
- Estas herramientas comparan la información del sistema con bases de datos de vulnerabilidades conocidas, como el National Vulnerability Database (NVD).

Revisión de la Configuración y Auditoría de Seguridad

- Revisar configuraciones de servidores, bases de datos, y otras tecnologías para asegurar que siguen las mejores prácticas de seguridad.
- Herramientas como Microsoft's Security Compliance Toolkit pueden ayudar en entornos Windows.

Uso de Listas de Control de Acceso y de Software de Seguridad

- Asegurarse de que el software de seguridad esté actualizado y configurado adecuadamente.
- Implementar listas de control de acceso para restringir quién puede acceder a qué en el sistema.

¿Preguntas?

Gracias.