Miércoles 24 de julio de 2013

El presente documento hace referencia al Marco de Seguridad que considerará el desarrollo de los sistemas en la iniciativa Salud Mesoamérica 2015. Este documento servirá como guía para el modelado de la arquitectura de software. Está basado en el Top 10 del Open Web Application Security Project (OWASP), un proyecto de código abierto dedicado a determinar y combatir las causas que hacen que el software sea inseguro. La comunidad OWASP está formada por empresas, organizaciones educativas y particulares de todo mundo. Juntos constituyen una comunidad de seguridad informática que trabaja para crear artículos, metodologías, documentación, herramientas y tecnologías que se liberan y pueden ser usadas gratuitamente por cualquiera.

Para hacer intrínseca a la seguridad en el desarrollo, se considerarán como partes esenciales en todos y cada uno de los Sprints de Scrum los conceptos de autenticación, administración de sesión, controles de acceso, validación de entradas, codificación de salidas (escapes a caracteres, encoding), criptografía, manejo de errores, loggin/out, protección de datos, comunicación segura y configuraciones de seguridad HTTP.

La OWASP lista los 10 principales fallos de seguridad de las plataformas en línea, junto con sus mecanismos para contrarrestar la vulnerabilidad. Es necesario establecer las políticas de seguridad en coordinación entre el administrador de la Base de datos, el administrador de Redes, el Arquitecto de Software y el equipo de Desarrollo.

Aquí el listado de vulnerabilidades

**1. Cross Site Scripting (XSS)**

El html generado por el cliente es ejecutado por el navegador web

\* Reflejado: Enlaces en correos, páginas web, etc.

\* Almacenado: Correo Web, foros, blogs, etc.

\* Inyección DOM: manipulación document.URL, document.location, etc.

Recomendaciones:

\* Validar y/o codificar todos los parámetros antes de incluirlos en páginas HTML.

\* Validar "Listas Blancas" (Whitelists): variables, usuarios, roles y privilegios aceptados, aprobados y reconocidos.

**2. Injection Flaws**

Los datos proporcionados por el usuario son enviados a un interpretador como parte de un comando o de un query, el más común es SQL injection.

Recomendaciones:

\* Validar "Listas Blancas" de lo que un usuario puede hacer

\* Revisar y validar variables introducidas por usuarios

\* Normalizar variables de entrada

\* Validación del lado del Cliente

\* Validación del lado del servidor

\* Restringir los tipos de datos aceptados

**3. Inclusión Insegura de archivos remotos**

Permite al atacante ejecutar código remoto, comprometiendo archivos de entrada. Aquellas aplicaciones que permiten la ejecución de archivos cargados son vulnerables.

Recomendaciones:

\* Uso de referencias indirectas (Otros servidores)

\* Validar "Listas Blancas" de lo que un usuario puede hacer

\* Diferenciar datos validados de los usuarios

\* Uso de mecanismos de aislamiento como chroot, jail, máquinas virtuales

\* Uso de mecanismos de lenguaje como tainting, allow\_url\_fopen

**4. Insecure Direct Object Reference**

La aplicación expone una referencia a un objeto interno como un directorio o una base de datos. Manipulación de parámetros para cambiar referencias y violar políticas de control. Las referencias a las llaves de las bases de datos están expuestas (index.php?id\_paciente=21).

Recomendaciones:

\* Evitar exponer referencias (ubicaciones o referencias internas a la estructura del servidor)

\* Validación con listas blancas

\* Verificar la autorización en los accesos

\* Usar índices o mapas de referencias (en lugar de http://www.some.com/app?file=nombreARchivo.doc usar http://www.some.com/app?file=1)

**5. Falsificación de Petición en Sitos Cruzados**

El navegador genera peticiones HTTP ocultas a recursos restringidos aprovechando autentificación implícita (autentificación http o login, Cookies, SSL, Ips, etc)

Recomendaciones:

\* Usar una cadena de autorización que no sea enviada automáticamente por el navegador

\* Eliminar cualquier vulnerabilidad de XSS en la aplicación

\* Añadir la petición a una variable de un sólo uso al URL y las formas, adicional a la sesión estándar

\* No usar método GET para requerir datos sensibles

**6. Fuga de Información y Manejo inadecuado de Errores**

La aplicación filtra información sensible sobre su configuración, diseño interno, datos privados etc. que pueden ser usados para hacer otros ataques. Dichas fugas provienen de los comentarios en el código fuente y de lo que reportan los mensajes de error (Como una cadena sql que no se ejecutó, o la referencia al disco duro del servidor de un archivo que no se encontró)

Recomendaciones:

\* Evitar la divulgación de mensajes de error detallados en el entorno de producción (MAS NO EN EL DE PRUEBAS)

\* Comprobar la aplicación con todo tipo de datos de entrada inválidos para analizar los mensajes de error generados

\* Deshabilitar o limitar los detalles mostrados sobre errores de capas especialmente sensibles: módulo de seguridad, Capa de Datos, SO, etc.

\* No usar gestores de error por defecto

\* Garantizar que los caminos de ejecución sensibles devuelven mensajes de error descriptivos mas no reveladores

**7. Débil protección de elementos de sesión y autenticación**

Fallo al proteger credenciales y cadenas (tokens) de sesión por fallas en los mecanismos de autenticación, debilidades en la administración de contraseñas, fallas en los tiempos de desconexión de las sesiones.

Recomendaciones:

\* Uso de SSL exclusivamente para todo acceso autenticado

\* Encriptar todas las credenciales y cadenas (tokens) para almacenarlos

\* Planificación cuidadosa de no exponer datos sensibles en URLs o registros; Utilizar un único mecanismo de autentificación; No usar direcciones IP, consultas al DNS o referrer headers para autenticación; cuidado en el envío de correos electrónicos con contraseñas.

\* Limitar o eliminar el uso de cookies para la autenticación o gestión de sesiones.

\* No aceptar id. de sesión nuevos, preestablecidos o inválidos en URLs o peticiones (Session Fixation Attacks, leer http://en.wikipedia.org/wiki/Session\_fixation)

\* Crear una nueva sesión tras la autentificación o cambio de nivel de privilegio

\* Proporcionar enlaces para desconectarse

\* Utilizar mecanismos de auto desconexión

\* Asegurarse que la liga Logout destruye todos los datos pertinentes

**8. Almacenamiento inseguro de funciones criptográficas**

Fallas al encriptar (o no encriptar at all) datos sensibles por usar algoritmos criptográficos propios, usar incorrectamente algoritmos fuertes, usar algoritmos ya considerados débiles (MD5, SHA-1, RC3, RC4) y uso de claves pre programadas o almacenadas.

Recomendaciones:

\* Sólo almacenar lo impresindible

\* Usar algoritmos provados (No crear nuevos) como AES, RSA para asimétricos, SHA-256 para hashings, (ver https://www.owasp.org/index.php/Guide\_to\_Cryptography)

\* No usar algoritmos débiles (MD5, SHA-1)

\* Llevar una gestión cuidadosa de las claves generándolas offline, almacenándolas con extremo cuidado y nunca transmitirlas por canales inseguros.

**9. Comunicaciones Inseguras**

La aplicación falla en encriptar el tráfico de la red.

Recomendaiones:

\* Usar SSL para conexiones autenticadas o que transmiten información sensible

\* Encriptar la comunicaciones en la infraestructura (Servidor, Base de datos, LDAP)

\* No permitir el uso de la aplicacion si no hay la certeza de un entorno seguro (si está deshabilitado temporalmente https, no abrir el sitio por http en lo que se reestablece el servicio)

**10. Falla para restringir el acceso mediante URLs**

La aplicación permite el acceso a funciones basándose en el URL

Recomendaciones:

\* Use de whilelists

\* Restringir el acceso a URLs y funciones en cada paso

\* Realizar pruebas de penetración

\* No asumir que los usuarios desconcen ciertas URLs