Tema 3a – La seguridad desde el inicio:

Introducción:

La seguridad debe introducirse en las **etapas iniciales del diseño**.

Posible resistencia: No hay tiempo disponible, no aporta valor comercial, requiere personal adicional, no hay garantías de ataque.

Mitos de seguridad:

1. Tenemos un firewall.

Frontera clara entre el mundo exterior y la compañía. El rol de un profesional de la seguridad era la configuración de la seguridad de red.

Caso 1: se elige el **personal de seguridad** del software de entre el equipo de seguridad en red. (Solución, implementar un firewall y el equipo de desarrollo "complaciente")

Caso 2: un **conferenciante comenta** que "la era del hacker de red está en decadencia". ("Hatemail" del público administrador de firewall, 70% de ataques en la capa de aplicación)

Sin fronteras: fronteras inexistentes, peligro del "cloud computing", el firewall no protege.

El enemigo interno: ataques dentro de la compañía, empleado, agente, "bombas lógicas".

Firewall (control perimetral), su lugar como **primera línea de defensa**, no único para proteger las **aplicaciones internas**.

Bueno: Filtrado de entrada, evitar denegación de servicios.

Malo: Filtrado de salida (DLP), robo de datos o ataques relacionados con datos.

2. Usamos SSL.

Protocolo que crea un túnel cifrado entre dos puntos. El sistema de certificados preinstalado **permite evitar el ataque MITM, evita que el firewall inspeccione el tráfico**. (Motorista)

Caso 1: Smak usó vulnerabilidades del sistema operativo y colocó sniffers en puntos clave.

Caso 2: No ofrece protección frente a ataques como inyección SQL o Cross-Site Scripting, hay ataques directos a la criptografía.

3. Tenemos IDPSs.

La mayoría de IDPS **proporciona alertas acerca de elementos desde inocuos hasta verdaderamente maliciosos.** La cantidad es tal, que peligros reales pueden quedar ocultos en los voluminosos logs. Análisis automatizado necesario, la calidad del sistema depende de la configuración de filtros y la monitorización de los logs. Necesario, complementan otras defensas (Controles perimetrales, antimalware, endurecimiento de aplicaciones).

4. Nuestro software no es accesible desde internet.

No tiene en cuenta ataques desde el interior. Nº de ataques externos e internos similar. Lo importante es el daño que puede causar. Falsa sensación de seguridad (Stuxnet, Mito los 80).

5. Nunca hemos sido comprometidos.

Debemos ser seguros puesto que no nos han atacado todavía, no hay que caer en la seguridad a través del FUD.

Microsoft indica como ley #1: Nadie piensa que le va a suceder algo malo hasta que le ocurre, cuando ocurre, el daño es devastador.

6. La seguridad no es "mi trabajo", es responsabilidad del proveedor.

Los proveedores de "cloud computing" tienen expectativas en sentido contrario. Proveedor 10% de sus recursos a la seguridad.

La actitud de "la seguridad no es cosa mía" produce aplicaciones con escasa seguridad. En caso de fallo de seguridad, es la empresa quien queda expuesta ante sus clientes y no el proveedor.

7. La seguridad aporta escasos beneficios al negocio.

Para una compañía que ha sido atacada de forma satisfactoria, la seguridad es un valor esencial. La realidad es que la seguridad añade valor en términos de permitir a la empresa continuar operaciones y generar el valor de negocio esperado

Si la web es atacada y "degradada", la empresa no puede continuar su actividad comercial y sufre pérdidas. La seguridad evita que esto ocurra.

El valor de la seguridad debe tomarse desde una perspectiva de ahorro además de una basada en el retorno a la inversión (ROI). Ahorro en reparación de vulnerabilidades y de reputación.

Resumen:

Necesidad: Ataques por "ego" y ataques por dinero.

Elementos: Motivación, oportunidad y medios. La mayoría de los programas de seguridad actuales se basan exclusivamente en herramientas ad-hoc. Prioridad y poca inversión.

Valor añadido: Es fiable (funciona como se espera), es resiliente (resiste abuso y ataques) y es recuperable (restauración de actividad comercial).

Menor posibilidad de error y posibilidad de publicar información sensible. Está disponible cuando se necesita. Se diseña bajo especificaciones funcionales y de seguridad. Es menos susceptible a fallos lógicos o semánticos y cumple las regulaciones establecidas.

Se ha modelado el riesgo y se conoce la superficie de ataque, su superficie de ataque relativa es reducida. Es seguro frente a ataques comunes. Ha sido auditado en busca de vulnerabilidades. Se ha instalado y configurado de forma adecuada. Se monitoriza y actualiza regularmente. Trata y elimina los datos de forma segura.

Tema 3b – SDL:

Evolución del cibercrimen:

Coste del cibercrimen en USA: Sobre $70B.

Ataques dirigidos a aplicaciones:

90% de vulnerabilidades son explotables de forma remota.

Evolución de la seguridad en MS:

2002-2003 -> "Windows security push" para Windows Server 2003

2004 -> El equipo directivo de seguridad en Microsoft acuerda exigir SDL para todos los productos que: estén expuestos a un riesgo significativo y/o, y procesen datos sensibles.

2005 -> SDL mejorado y Windows Vista es el primer SO en pasar por el ciclo SDL completo.

Ahora -> Optimización mediante retroalimentación, análisis y automatización, Evangelización de SDL a la comunidad de desarrollo.

¿Qué aplicaciones deben seguir SDL?:

Producto utilizado habitualmente o **implementado en empresas u organizaciones**, que almacene o comunique datos sensibles o personales (identificativos), **en contacto con Internet** u otras redes, que acepte y/o procese **datos provenientes de una fuente no autentificada**, que interprete tipos de **fichero no protegidos** (no limitados a administradores), que contenga controles ActiveX y/o COM, **de Microsoft, MSN y Live.com que son accedidos por clientes externos.**

EL SDL:

Entregar software seguro requiere: (**Compromiso directiva** → SDL es política obligatoria en Microsoft desde 2004). **Mejoras continuas del proceso** → ciclo de 6 meses.

- ### Requisitos Pre-SDL: Entrenamiento.

Evaluar el conocimiento de la organización respecto a seguridad y privacidad; establecer el programa de entrenamiento de la manera necesaria.

(Hitos respecto al entrenamiento, cubrir el contenido y asistencia)

- ### Fase 1: Requisitos.

Oportunidad para considerar la seguridad desde el inicio.

(Identificar encargados, requisitos de seguridad y privacidad, Asesor de Seguridad, uso de sistema de seguimiento de bugs, asignar trabajo, definir y documentar objetivos)

- Fase 2: Diseño.

Definir y documentar la arquitectura y componentes críticos de seguridad.

(Técnicas de diseño, documentar la superficie de ataque y limitar en los valores por defecto, definir características de seguridad adicionales particulares al producto, modelado de amenazas y requisitos específicos para servicios online)

- Fase 3: Implementación.

Revisión completa – para determinar procesos, documentación y herramientas necesarias para garantizar una instalación y operación seguras.

(Especificación de herramientas y opciones de compilación aprobadas, análisis estático, APIs prohibidas, uso de protecciones del SO "en profundidad" y considerar otras recomendaciones)

- Fase 4: Verificación.

Inicio tan pronto como sea posible, al tener el código completo.

(Iniciar la planificación de respuesta de seguridad – incluir planes de respuesta para informes de vulnerabilidad, reevaluar la superficie de ataque, Fuzz testing y realizar un "security push")

- Fase 5: Entrega.

Plan de respuesta.

Definición clara de la política de soporte – de acuerdo con las políticas corporativas de MS.

Revisión Final de Seguridad.

Verificar el cumplimiento de los requisitos SDL y que no hay vulnerabilidades de seguridad conocidas. (Factor para determinar si la aplicación debe ser entregada o no, evitando cajón)

Archivado.

El plan de respuesta de seguridad está completado. (Documentación del cliente está actualizada, completar las autorizaciones finales en Checkpoint Express y archivar modelo de amenaza RTM)

- Requisito Post-SDL: Respuesta.

"Planifica el trabajo, trabaja el plan…" (Ejecución de las tareas de respuesta establecidas durante las fases de planificación de respuesta de seguridad y entrega)

Conclusiones:

Los ataques van dirigidos a las **aplicaciones**.

El SDL consiste en **incorporar seguridad** en el desarrollo de software y en la cultura corporativa.

Se han obtenido **resultados medibles** en el caso del software de Microsoft.

Microsoft se ha comprometido a que el SDL esté **disponible y sea accesible**.

Fuzz Testing:

Consiste en **introducir datos malformados y/o erróneos en la aplicación comprobando la reacción de esta**. Si la aplicación falla es que se ha encontrado una vulnerabilidad potencial. Se trata de un complemento, no se utiliza como reemplazo de otras técnicas de evaluación.

Ventajas: Vulnerabilidades identificadas de naturaleza severa, se puede automatizar fácil.

Desventajas: No puede identificar vulnerabilidades que no causan una excepción. (Filtración de información, fallo en la criptografía)

Tipos de Fuzz Testing:

Aleatorio: Se introducen datos en la aplicación que cambian de forma puramente aleatoria.

Inteligente: Se modifican valores específicos en función de la experiencia previa y/o el comportamiento esperado.

Realización de Fuzz Tests:

1. Determinar los puntos de entrada a la aplicación.

2. Ordenar dichos puntos de entrada en función del privilegio y la accesibilidad.

3. Crear e introducir datos malformados en la aplicación para evaluar los puntos de entrada que están en riesgo. Intentar automatizar el proceso en la medida de lo posible.

4. Analizar cualquier fallo, excepción o error inesperado o no gestionado debidamente para identificar vulnerabilidades severas.

5. Reparar y volver a comprobar.

Tema 3c – Fundamentos del Diseño Seguro:

Reducción de la Superficie de Ataque:

Cualquier parte de una aplicación que es accesible a un humano u otro programa. Cada una de estas puede ser potencialmente explotada por un usuario malicioso.

Minimizar el número de **puntos expuestos en la superficie de ataque** que un atacante podría descubrir y explotar.

Análisis de la Superficie de Ataque:

Paso 1: Definirlos puntos de entrada. (Entradas/Salidas de ficheros/red)

Paso 2: Priorizar los puntos de entrada. (Autentificado o anónimo, administrativo, red/local...)

Proceso iterativo, para toda funcionalidad hay que analizar las subfuncionalidades, hay que restringir el acceso a cada funcionalidad al máximo posible

No consiste únicamente en apagar cosas:

Menor superficie: apagado por defecto, socket cerrado, TCP, acceso autentificado, activo bajo demanda, acceso a nivel de usuario, accesible por red local, ejecución como usuario (user space), valores por defecto definidos por el usuario, código reducido y controles de acceso fuertes.

Ejemplos de Reducción de SA:

Windows: Llamada de procedimiento remoto (RPC) autentificada, firewall activo por defecto.

Internet Inf. Services: desactivado, ejecución como servicio de red y ficheros estáticos por defecto.

SQL Server: Procedimiento xp_cmdshell, CLR y COM y conexiones remotas deshabilitadas por defecto.

Visual Studio: ServidorWeb y SQL Server Express en localhost únicamente.

Privacidad Básica:

Privacidad: permitir a los usuarios controlar el uso, recolección y distribución de su información personal.

Seguridad: establecer medidas de protección para garantizar la integridad, disponibilidad y confidencialidad de la información.

La privacidad y seguridad son **factores clave en la construcción de aplicaciones de confianza.**

Es posible tener un sistema seguro donde **NO** se preserve la privacidad de los usuarios.

N**ecesario la seguridad para que exista la privacidad**.

Comportamiento y Controles:

Se dirige a menores: COPPA	Modifica el sistema: CFAA
Transfer información personal sensible: GLBA, HIPAA	Monitoreo continuo: Anti-Sypyware
Transfer información personal no sensible: UE, FTC	Transferencia anónima: Block Install

Modelado de Amenazas:

Un **proceso para entender las amenazas** que puede sufrir una aplicación. (Visión general)

Amenazas: Lo que un usuario malicioso puede intentar para comprometer un sistema.

Vulnerabilidades: Una forma concreta en la que una amenaza es explotable, como un error de programación por ej.

Defensa en profundidad:

Asumir que el software y el hardware fallarán en algún momento.

La mayoría de las aplicaciones actuales son comprometidas al romper una, habitualmente única, capa de seguridad (firewall).

La defensa en profundidad proporciona seguridad adicional a la aplicación, añadiendo capas de defensa.

Minimización de privilegio:

Asumir que todas las aplicaciones pueden ser y serán comprometidas.

Si una aplicación es comprometida, el daño potencial que puede causar el atacante es contenido y minimizado convenientemente.

El acceso mínimo es el del usuario, elevar los privilegios sólo cuando sea necesario, para posteriormente liberar dichos privilegios elevados cuando se han satisfecho los requisitos.

Valores por defecto seguros:

Entrega de aplicaciones con configuraciones más seguras por defecto.

Ayuda a que los clientes tengan una experiencia más segura al instalar la aplicación, no tras una configuración extensa.

Se deja al usuario la posibilidad de reducir la seguridad o los niveles de privacidad.

Configuración por defecto:

Firewall activo, exigir última versión de SSL, Firewall activo, sesiones de usuario autentificadas complejidad de contraseña forzada y almacenaje de hashes de contraseña.

Análisis de código:

Herramientas software que **analizan la implementación de la aplicación** para comprobar que siguen las prácticas recomendadas.

Pueden **reducir en gran medida los costes de ingeniería**.

Análisis estático de código fuente:

Herramientas software que **analizan las implementaciones en código fuente** en busca de posibles mejoras.

Entradas: código fuente legible, como ficheros C (*.c), C++ (*.cpp, *.cc) o C# (*.cs).

Ventajas: Facilidad de diagnóstico de problemas y tecnología más madura.

Análisis de código binario:

Herramientas software que **analizan código compilado** o una versión en binario del código fuente en busca de posibles mejoras.

Entradas: código máquina o ficheros binarios, como ficheros ejecutables (*.exe) y librerías (*.dll)

Ventaja: La herramienta de análisis de código binario puede ver el código binario en sí mismo (resultado final)

Análisis de código: pros y contras:

Pros: Ayuda a escalar el proceso de revisión de código y ayuda a cumplir políticas de programación segura.

Contras: Falsos positivos, falsos negativos, dependiente del lenguaje, problemas únicamente a nivel de código.