INFORME

OWASP

Javier Godia
Daniel Padilla
Tomás López

Índice

Introducción

El presente documento define un conjunto de prácticas comunes de codificación de software, tecnológicamente agnósticas, en formato de lista de verificación con el fin de ser integrado al ciclo de desarrollo. La implementación de estas prácticas mitiga las vulnerabilidades más comunes del software.

En términos generales, es mucho menos costoso construir software seguro que corregir problemas de seguridad cuando ha sido completado, sin mencionar los costos quepueden estar asociados a un quiebre en la seguridad.

Asegurar recursos críticos de software se ha vuelto más importante que nunca debido a que el foco de los atacantes se ha desplazado hacia la capa de aplicación**.** Según el informe de la empresa tecnológica NTT, de 2021(Global Threat Intelligence Report) determina que los ataques contra aplicaciones web se disparan hasta suponer el 67 % del total de los ataques.

Utilizando la presente guía, el equipo de desarrollo debería comenzar evaluando el grado de madurez de su ciclo de desarrollo de software desde el punto de vista de la seguridad, así como el nivel de conocimiento de sus miembros. Dado que esta guía no cubre los detalles de implementación de las prácticas de codificación, los desarrolladores deben poseer previamente los conocimientos suficientes o bien tener disponibles recursos que los guíen. Esta guía provee prácticas de codificación que pueden ser traducidas en requisitos de codificación sin la necesidad que el desarrollador posea un entendimiento profundo de vulnerabilidades de seguridad y exploits. Sin embargo, otros miembros del equipo de desarrollo deberían de poseer el entrenamiento adecuado, recursos y herramientas para ser responsables y validar que el diseño e implementación del sistema es seguro.

En el apéndice B se presenta un glosario de los términos más importantes utilizados en este documento, incluyendo los cabezales de sección y las palabras resaltadas con itálicas.

Visión General de la Seguridad en el Software y sus Principales Riesgos

La construcción de software seguro requiere una comprensión básica de los principios de seguridad. Si bien escapa al objetivo de esta guía una revisión exhaustiva de estos, se proporciona una visión general y rápida de los mismos.

El objetivo de la seguridad en el software es el de mantener la *confidencialidad*,*integridad* y *disponibilidad* de los recursos de información de modo de permitir el desarrollo exitoso de negocios. Este objetivo se consigue con la implementación de *controles de seguridad*. La presente guía pone foco en los controles técnicos específicospara *mitigar* la presencia de *vulnerabilidades* de ocurrencia frecuentes en el software. Sibien el foco principal está en las aplicaciones web y la infraestructura que las soporta, lamayoría de los consejos puede aplicarse en cualquier plataforma de desarrollo de software.

Es de gran utilidad comprender qué se entiende por un riesgo, con el fin de proteger al negocio de riesgos inaceptables derivados de su dependencia en el software. Un riesgo es una combinación de factores que amenazan al éxito del negocio. Puede ser definido conceptualmente como: un *agente amenazante* que interactúa con el *sistema*, que puede tener una *vulnerabilidad* a ser *explotada* de forma de generar un *impacto*.

Si bien este puede resultar un concepto abstracto, se lo puede pensar de esta forma: un ladrón de coches (agente amenazante) recorre un estacionamiento verificando los vehículos (el sistema) en busca de una puerta sin trabar (la vulnerabilidad) y cuando encuentra uno, abre la puerta (situación a ser explotada) y toma algo de dentro del mismo (el impacto). Todos estos factores juegan un rol en el desarrollo de software seguro.

Existe una diferencia fundamental entre el enfoque que adopta un equipo de desarrollo y el que toma alguien que está realizando un ataque sobre la aplicación. Un equipo de desarrollo adopta un enfoque basado en lo que tiene intenciones de realizar. En otras palabras, están diseñando una aplicación para que realice tareas específicas basándose en los requerimientos funcionales y casos de uso que han sido documentados. Un atacante, por otra parte, está más interesado en lo que puede llegar a hacerse con esa aplicación y opera bajo el principio de “cualquier acción que no haya sido denegada de forma expresa, está permitida”.

Para hacer frente a esto, se deben integrar algunos elementos adicionales en las etapas tempranas del ciclo de desarrollo de software. Estos nuevos elementos son los *requerimientos de seguridad* y los *casos de abuso*. Esta guía fue creada para ayudar a identificar requerimientos de seguridad de alto nivel y hacer frente a muchos escenarios de abuso comunes.

Es importante que los equipos de desarrollo web entiendan que los controles desde el lado del cliente, tales como la validación de lo que ingresa el cliente, campos ocultos y controles en la interfaz (por ejemplo: menús desplegables y botones de opción) brindan poco si acaso algún beneficio desde el punto de vista de la seguridad. Un atacante puede utilizar herramientas tales como proxies del lado del cliente (por ejemplo: WebScarab o Burp de OWASP) o herramientas de capturas de paquetes de red (como WireShark) para analizar el tráfico de la aplicación y enviar solitudes hechas a medida, sin siquiera

pasar por la interfaz. Además de esto, Applets de Java y otros objetos del lado delcliente pueden ser descompilados y analizados en busca de fallas.

Las fallas de seguridad en el software pueden introducirse en cualquiera de las etapas del ciclo de desarrollo del software, derivados de:

- No identificar requerimientos de seguridad desde el inicio.
- Crear diseños conceptuales que contengan errores en la lógica.
- Utilizar prácticas débiles de codificación que introduzcan vulnerabilidades desde el punto de vista técnico.
- Implantación del software de forma inapropiada.
- Introducción de fallas durante el mantenimiento o actualización del producto.

Es importante además entender que las vulnerabilidades del software pueden escapar los límites del software mismo. Dependiendo de la naturaleza del software, la vulnerabilidad y la infraestructura que da soporte, el impacto del éxito en la intromisión puede comprometer a cualquiera o todas de las siguientes:

- El software y su información asociada.
- Los sistemas operativos de los servidores asociados.
- La base de datos del backend.
- Otras aplicaciones en el entorno compartido.
- El sistema del usuario.
- Otro software con el que interactúe el usuario.

Lista de Verificación de Prácticas de Codificación Segura

Validación de entradas

- ☐ Realizar todas las validaciones de datos en un sistema confiable (por ejemplo: el servidor).
- ☐ Identificar todas las fuentes de datos y clasificarlos como confiables o no confiables. Validar todos los datos provenientes de fuentes no confiables (por ejemplo: bases de datos, secuencias de archivo, etc.).
- ☐ Debería existir una rutina de validación de datos de entrada centralizada para la aplicación.
- ☐ Especificar sets de caracteres apropiados, tales como UTF-8, para todas las fuentes de entrada.
- ☐ Codificar los datos a un set de caracteres común antes de validar (*Canonicalización*).
- ☐ Todas las fallas en la validación deber terminar en el rechazo del dato de entrada.
- ☐ Determinar si el sistema soportará sets de caracteres UTF-8 extendidos y de ser así, validarlos luego de terminada la decodificación del UTF-8.
- ☐ Validar todos los datos brindados por el cliente antes de procesarlos, incluyendo todos los parámetros, URLs y contenidos de cabeceras HTTP (por ejemplo nombres de Cookies y valores). Asegurarse de incluir post backs automáticos desde JavaScript, Flash u otro código embebido.
- ☐ Verificar que los valores de la cabecera tanto en solicitudes como en respuestas contengan solo caracteres ASCII.
- ☐ Validar datos redireccionados (un atacante puede enviar contenido malicioso directamente en el destino de la redirección, eludiendo entonces la lógica de la aplicación y cualquier otra validación realizada antes de la redirección).
- ☐ Validar tipos de datos no esperados.
- ☐ Validar rangos de datos.
- ☐ Validar largos de datos.
- ☐ Validar toda entrada con una lista “blanca” que contenga una lista de caracteres aceptados, siempre que sea posible.
- ☐ Si es necesario, permitir el ingreso de algún *carácter considerado peligroso*, asegúrese de implementar controles adicionales tales como la codificación de la salida, API de seguridad y el registro del uso de tales datos a lo largo de laaplicación.

Entre los ejemplos de caracteres peligrosos podemos encontrar: < > ” ’

% ( ) & + \ / \’ \”.

- ☐ Si su rutina estándar de validación no contempla el ingreso de los ejemplos de datos anteriormente ejemplificados, entonces deberán verificarse de forma puntual.
- ☐ Compruebe si hay bytes nulos (%00).
- ☐ Compruebe si hay caracteres de nueva línea (%0d, %0a, \r, \n).
- ☐ Compruebe si hay caracteres de alteraciones de ruta “punto, punto, barra (../ o ..\). En los casos en que se soportan sets de caracteres UTF-8 extendidos, implemente representaciones alternativas tales como: %c0%ae%c0%ae/ (utilice la *canonicalización* como forma de implementar la doble codificación u otras formas de ofuscación de ataques).

Codificación de salidas

- ☐ Realice toda la codificación en un ambiente seguro (por ejemplo: el servidor).
- ☐ Utilice una rutina probada y estándar para cada tipo de codificación de salida.
- ☐ *Contextualice la codificación de salida* de todos los datos devueltos por el cliente que se originen desde fuera de la *frontera de confianza* de la aplicación. La *codificación de entidades HTML* es un ejemplo, aunque no sea suficiente en todos los casos.
- ☐ Codifique todos los caracteres salvo que sean reconocidos como seguros por el interpretador al que están destinados.
- ☐ *Sanitice* contextualmente todos las salidas de datos no confiables hacia consultas SQL, XML y LDAP.
- ☐ *Sanitice* todas las salidas de datos no confiables hacia un comando del sistema operativo.

Administración de autenticación y contraseñas

- ☐ Requerir autenticación para todos los recursos y páginas excepto aquellas específicamente clasificadas como públicas.
- ☐ Todos los controles de autenticación deben ser efectuados en un sistema en el cual se confíe. (por ejemplo: el servidor).
- ☐ Establecer y utilizar servicios de autenticación estándares y probados cuando sea posible.

- ☐ Utilizar una implementación centralizada para todos los controles de autenticación, incluyendo librerías que llamen a servicios externos de autenticación.

- ☐ Segregar la lógica de la autenticación del recurso solicitado y utilizar redirección desde y hacia el control centralizado de autenticación.

- ☐ Todos los controles de autenticación deben fallar de una forma segura.

- ☐ Todas las funciones administrativas y de administración de cuentas deben ser al menos tan seguras como el mecanismo primario de autenticación.

- ☐ Si la aplicación administra un almacenamiento de credenciales, se debe asegurar que únicamente se almacena el hash con sal (salty hash) de las contraseñas y que el archivo/tabla que guarda las contraseñas y claves solo puede ser escrito por la aplicación (si es posible, **no utilizar el algoritmo de hash MD5**). Sería recomendable el uso de algoritmos, como por ejemplo SHA-256. Evitar la codificación de contraseñas en base64 o similares.

- ☐ El hash de las contraseñas debe ser implementado en un sistema en el cual se confíe. (por ejemplo: el servidor).

- ☐ Validar los datos de autenticación únicamente luego haber completado todos los datos de entrada, especialmente en implementaciones de *autenticación secuencial*.

- ☐ Las respuestas a los fallos en la autenticación no deben indicar cual parte de la autenticación fue incorrecta. A modo de ejemplo, en lugar de "usuario invalido" o "contraseña invalida", utilizar "usuario y/o contraseña inválidos" en ambos casos. Las repuestas a los errores deben ser idénticas tanto a nivel de lo desplegado como anivel de código fuente.

- ☐ Utilizar autenticación para conexiones a sistemas externos que involucren información o funciones sensibles.

- ☐ Las credenciales de autenticación para acceder a servicios externos a la aplicación deben ser encriptados y almacenados en ubicaciones protegidas en un sistema en el cual se confíe (ejemplo: el servidor). El código fuente NO es una ubicación segura.

- ☐ Utilizar únicamente pedidos del tipo HTTPS.

- ☐ Utilizar únicamente conexiones encriptadas o datos encriptados para el envío de contraseñas que no sean temporales (por ejemplo: correo encriptado). Contraseñas temporales como aquellas asociadas con reseteos por correo electrónico, pueden ser una excepción.

- ☐ Hacer cumplir por medio de una política o regulación los requerimientos de complejidad de la contraseña. Las credenciales de autenticación deben ser suficientes como para resistir aquellos ataques típicos de las amenazas en el entorno del sistema. Ej: obligar el uso de combinaciones de caracteres numéricos/alfanuméricos y/o caracteres especiales.

- ☐ Hacer cumplir por medio de una política o regulación los requerimientos de longitud de la contraseña. Comúnmente se utilizan ocho caracteres, pero dieciséis es mejor, adicionalmente considerar el uso de frases de varias palabras.

- ☐ No se debe desplegar en la pantalla la contraseña ingresada. A modo de ejemplo, en formularios web, utilizar el tipo de entrada “password” (input type “password”).

- ☐ Deshabilitar las cuentas luego de un número establecido de intentos inválidos de ingreso al sistema. A modo de ejemplo, tres intentos fallidos es lo común. La deshabilitación de la cuenta debe ser por un periodo de tiempo suficiente como para desalentar una inferencia de credenciales por fuerza bruta, pero no tan alto como para permitir un ataque de negación de servicio.

- ☐ El cambio y reseteo de contraseñas requieren los mismos niveles de control como aquellos asociados a la creación y autenticación de cuentas.

- ☐ Las preguntas para el reseteo de contraseñas deben contemplar un amplio rango de respuestas aleatorias. A modo de ejemplo evitar peguntas muy básicas, que tengan una respuesta obvia y muy común.

- ☐ Si se utiliza un reseteo por correo electrónico, únicamente enviar un link o contraseñas temporales a casillas previamente registradas en el sistema.

- ☐ Las contraseñas y links temporales deben tener un corto periodo de validez.

- ☐ Forzar el cambio de contraseñas temporales luego de su utilización.

- ☐ Notificar a los usuarios cada vez que se produce un reseteo de contraseña.

- ☐ Prevenir la reutilización de contraseñas.

- ☐ Las contraseñas deben tener al menos un día de antigüedad antes de poder ser cambiadas, de forma de evitar ataques de reutilización de contraseñas.

- ☐ Hacer cumplir por medio de una política o regulación los requerimientos de cambio de contraseña. Los sistemas críticos pueden requerir cambios más frecuentes que otros sistemas. El tiempo entre cada reseteo debe ser controlado administrativamente.

- ☐ Deshabilitar la funcionalidad de “recordar” campos de contraseñas.

- ☐ El último acceso (fallido o exitoso) debe ser reportado al usuario en su siguiente acceso exitoso.

- ☐ Implementar un monitoreo para identificar ataques a múltiples cuentas utilizando la misma contraseña. Este patrón de ataque es utilizado parar superar bloqueos estándar cuando los nombres de usuario pueden ser obtenidos o adivinados dealguna forma.

- ☐ Cambiar todos los usuarios y contraseñas por defecto o deshabilitar las cuentas asociadas.

- ☐ Re autenticar usuarios antes de la realización de operaciones críticas.
- ☐ Utilizar *autenticación multi-factor* para las cuentas más sensibles o de mayor valor.
- ☐ Si se utiliza un código de una tercera parte para la autenticación, inspeccionarlo minuciosamente para asegurar que no se encuentre afectado por cualquier código malicioso.

Administración de sesiones

- ☐ Utilizar los controles del servidor o del framework para la administración de sesiones. La aplicación solo debe reconocer estos identificadores como válidos.
- ☐ La creación de identificadores de sesión solo debe ser realizada en un sistema en cual se confíe (por ejemplo: el servidor).
- ☐ Los controles de administración de sesiones deben utilizar algoritmos que generen identificadores suficientemente aleatorios.
- ☐ Definir el dominio y ruta para las cookies que contienen identificadores de sesión autenticados con un valor apropiadamente estricto para el sitio.
- ☐ La función de logout debe terminar completamente con la sesión o conexión asociada.
- ☐ La función de logout debe estar disponible en todas las páginas protegidas por autenticación.
- ☐ Establecer un tiempo de vida de la sesión lo más corto posible, balanceando los riesgos con los requerimientos del negocio. En la mayoría de los casos, nunca debería ser superior a varias horas.
- ☐ Deshabilitar logeos persistentes y efectuar finalizaciones periódicas de sesiones, incluso cuando la sesión se encuentra activa.
- ☐ Si una sesión fue establecida antes del login, cerrar dicha sesión y establecer una nueva luego de un login exitoso.
- ☐ Generar un nuevo identificador de sesión luego de cada re autenticación.
- ☐ No permitir logeos concurrentes con el mismo usuario.
- ☐ No exponer identificadores de sesión en URLs, mensajes de error ni logs. Los identificadores de sesión solo deben ser ubicados en la cabecera de la cookie HTTP. A modo de ejemplo, no transmitir el identificador de sesión como un parámetro GET.
- ☐ Proteger la información sobre las sesiones del lado del servidor implementando los controles de acceso apropiados.
- ☐ Generar un nuevo identificador de sesión y desactivar el anterior de forma periódica.

De esta forma se pueden mitigar algunos escenarios de robo de sesiones donde el identificador se compromete.

- ☐ Generar un nuevo identificador de sesión si la seguridad cambia de HTTP a HTTPS, como puede suceder durante la autenticación. Dentro de la aplicación esrecomendable usar siempre HTTPS en lugar de cambiar entre HTTP y HTTPS.

- ☐ Manejo de sesión complementario para operaciones sensible del lado del servidor, como pueden ser: gestión de cuentas o utilizando tokens o parámetros por sesión. Este método puede ser utilizado para prevenir ataques de Cross Site Request Forgery Attacks (*Falsificación de petición en sitios cruzados*, conocido por sus siglas CSRF).

- ☐ Manejo de sesión complementario para operaciones sensible o críticas utilizando tokens o parámetros por pedido (per request) en lugar de por sesión.

- ☐ Configurar el atributo "seguro" para las cookies transmitidas sobre una conexión TLS.

- ☐ Configurar las cookies con el atributo HttpOnly, salvo que se requiera específicamente scripts del lado del cliente en la aplicación, para leer o configurar una cookie.

Control de Acceso

- ☐ Utilizar únicamente objetos confiables del sistema. (por ejemplo: objetos de sesión del servidor para la toma de decisiones de autorización).

- ☐ Utilizar un único componente para el chequeo de autorizaciones para todo el sitio.

  Esto incluye librerías que llamen a servicios de autorización externos.

- ☐ Los controles de acceso en caso de falla, deben actuar en forma segura.

- ☐ Denegar todos los accesos en caso de que la aplicación no pueda acceder a la información de configuración de seguridad.

- ☐ Separar lógica privilegiada de otro código de la aplicación.

- ☐ Restringir acceso a ficheros u otros recursos, incluyendo aquellos fuera del control directo de la aplicación, únicamente a usuarios autorizados.

- ☐ Restringir el acceso a URLs protegidas, solo a usuarios autorizados.

- ☐ Restringir el acceso a funciones protegidas, solo a usuarios autorizados.

- ☐ Restringir las referencias directas a objetos, solo a usuarios autorizados.

- ☐ Restringir el acceso a servicios, solo a usuarios autorizados.

- ☐ Restringir el acceso a información de la aplicación, solo a usuarios autorizados.

- ☐ Restringir el acceso a usuario, atributos y política de información utilizada por los controles de acceso.
- ☐ Restringir el acceso a información relevante de la configuración, solo a usuarios autorizados.
- ☐ Las reglas de control de acceso implementadas del lado del servidor y en la capa de presentación, deben coincidir.
- ☐ Si existen *datos de estado* que deben ser guardados en el cliente, use cifrado y chequeos de integridad del lado del servidor para poder evaluar el estado.
- ☐ Requerir flujos de aplicación que cumplan con las reglas del negocio.
- ☐ Limitar el número de transacciones que un usuario común o un dispositivo puede desarrollar en un cierto período de tiempo.
- ☐ Utilizar el header "referer" solo como un chequeo complementario. Nunca debe ser utilizado como chequeo de autorización, ya que es posible modificarlo.
- ☐ Si sesiones de autenticación extensas son permitidas, periódicamente hay que re-validar la autorización de los usuarios y asegurar que sus privilegios no han sido modificados.
- ☐ Implementar auditorías de cuentas y requerir que se deshabiliten las contraseñas de las cuentas.
- ☐ La aplicación debe permitir deshabilitar y terminar cuentas una vez que se termina la autorización (Cambio de rol, estatus de empleo, etc).
- ☐ Cuentas de servicio o cuentas soportando conectividad deben tener el mínimo privilegio.
- ☐ Crear una política de control de acceso para documentar las reglas del negocio de la aplicación, los tipos de datos, criterios para autorización de acceso y los controles asociados para otorgarlos y controlarlos. Esto incluye la identificación de accesos requeridos tanto para los datos como para los sistemas.

Prácticas Critpográficas

- ☐ Todas las funciones de criptografía de la aplicación deben ser implementadas en sistemas de confiables (por ejemplo: el servidor).
- ☐ Proteger secretos maestros (master secrets) del acceso no autorizado.
- ☐ Los módulos de criptografía deberían en caso de falla, fallar en forma segura.
- ☐ Todos los números aleatorios, nombre aleatorios, GUIDs, y frases aleatorias, deberían generarse utilizando módulos aprobados para su generación.

- ☐ Establecer y utilizar una política y un proceso de cómo manejar las claves criptográficas.

Manejo de errores y Logs

- ☐ No difundir información sensible en respuestas de error, incluyendo detalles del sistema, identificadores de sesión o información de la cuenta.
- ☐ Utilizar manejadores de errores que no muestren información de debugging o de memoria.
- ☐ Implementar mensajes de error genéricos y utilizar páginas de error adaptadas.
- ☐ La aplicación debería manejar los errores de la aplicación y basarse en la configuración del servidor.
- ☐ Liberar espacio de memoria en cuanto una condición de error ocurra.
- ☐ L a lógica para la gestión de errores debe estar asociada a que los controles de seguridad no permitirán acceso por defecto.
- ☐ Todos los controles de logging deben estar implementados en sistemas confiables.
- ☐ El logging de controles de acceso debe incluir tanto los casos de éxito como de falla.
- ☐ Asegurar que los *datos del registro de log* contengan información importante.
- ☐ Asegurar que los logs de entrada que incluyen información no segura, no serán ejecutados en interfaces o aplicativos.
- ☐ Restringir el acceso a los logs, solo a personal autorizado.
- ☐ Utilizar una rutina centralizada para todas las operaciones de logging.
- ☐ No guardar información sensible en logs, incluyendo detalles innecesarios del sistema.
- ☐ Asegurar que existen mecanismos para conducir un análisis de los logs.
- ☐ Registrar en un log todas las fallas de validación.
- ☐ Registrar en un log todos los intentos de autenticación, en particular los fallidos.
- ☐ Registrar en un log todas las fallas en los controles de acceso.
- ☐ Registrar en un log todos los eventos de intento de evasión de controles, incluyendo cambios en el estado de la información no esperados.
- ☐ Registrar en un log todos los intentos de conexión con tokens inválidos o vencidos.

- ☐ Registrar en un log todas las excepciones del sistema.
- ☐ Registrar en un log todas las funciones administrativas, incluyendo cambios en la configuración de seguridad.
- ☐ Registrar en un log, todas las fallas de conexión de TLS.
- ☐ Registrar en un log las fallas de los módulos criptográficos.
- ☐ Utilizar una función de hash para validar la integridad de los logs.

Protección de datos

- ☐ Implementar el mínimo privilegio, restringir el acceso de los usuarios solamente a la funcionalidad, datos y sistemas de información que son necesarios para realizar sus tareas.
- ☐ Proteger todos los almacenamientos temporales (en memoria o archivo -"cached") de los datos sensibles guardados en el servidor de los accesos no autorizados y eliminar todos los archivos y memoria de trabajo temporal tan pronto como no sean requeridos.
- ☐ Encriptar toda la información altamente sensible almacenada, como por ejemplo: datos para la verificación de la autenticación, inclusive también en el servidor. Siempre utilice algoritmos de encriptación que hayan sido chequeados y con buenos antecedentes. Vea "Prácticas Criptográficas" para obtener información adicional.
- ☐ Proteja el código fuente del servidor de forma de que no pueda ser descargado por el usuario.
- ☐ Remueva los comentarios en el código de producción que sea accesible por el usuario que puedan revelar información sobre los servidores o información sensible.
- ☐ Remueva cualquier aplicación que no sea necesaria y la documentación de los sistemas que pueda revelar información útil para los atacantes.
- ☐ Deshabilite las funcionalidades de completar automáticamente en aquellos formularios que contienen información sensible, incluyendo la autenticación.
- ☐ Deshabilite el almacenamiento temporal del lado del cliente de páginas que contienen información sensible.
- ☐ La aplicación debe de soportar la eliminación de datos sensibles cuando los datos ya no son requeridos (por ejemplo: información personal o ciertos datos financieros).
- ☐ Implementar controles de accesos apropiados para los datos sensibles almacenados en el servidor. Esto incluye memoria temporal, archivos temporales y datos que solamente puedan ser accedidos por usuarios específicos del sistema

Seguridad en las comunicaciones

- ☐ Implemente encriptación para todas las transmisiones de información sensible. Esto debería incluir TLS para proteger la conexión y se puede combinar con una encriptación discreta de archivos sensibles en conexiones que no estén basadas en HTTP.
- ☐ Los certificados TLS deben de ser válidos y contener el nombre de dominio correcto, no deben estar expirados y deberán ser instalados con los certificados intermedios si son requeridos.
- ☐ Las conexiones TLS que fallen no deben de transformarse en una conexión insegura.
- ☐ Utilizar conexiones TLS para todo el contenido que requiera acceso autenticado y para todo otro tipo de información sensible.
- ☐ Utilizar TLS para las conexiones a sistemas externos que involucren funciones oinformación sensible.
- ☐ Utilizar una única implementación estándar de TLS que se encuentre configurada correctamente.
- ☐ Especificar los caracteres de codificación para todas las conexiones.

Configuración de los sistemas

- ☐ Asegurarse de que los servidores, los frameworks y los componentes del sistema están corriendo la última versión aprobada.
- ☐ Asegurar que los servidores, los frameworks y los componentes del sistema están actualizados con todos los parches emitidos para las versiones en uso.
- ☐ Deshabilitar el listado de directorio.
- ☐ Restringir el servidor web, los procesos y las cuentas de servicios con el mínimo privilegio posible.
- ☐ Cuando ocurra una excepción, se debe de fallar seguro.
- ☐ Remover todas las funcionalidades y archivos que no sean necesarios.
- ☐ Remover código de testeo o cualquier funcionalidad que no sea tenida en cuenta en producción, previo a realizar la puesta en producción.
- ☐ Prevenir la revelación de la estructura de directorios en el archivo robots.txt colocando directorios que estén disponibles para el índice público en un directorio raíz aislado. Luego “Deshabilitar” el directorio raíz en el archivo robots.txt en vez de “Deshabilitar” cada directorio individual.
- ☐ La configuración de seguridad de donde se encuentra almacenada la aplicación debe de ser listada en un formato legible para su auditoría.
- ☐ Implementar un Sistema de Manejo de Activos y registrar los componentes del sistema en él.
- ☐ Aislar los ambientes de desarrollo de los ambientes de producción y permitir el acceso solamente a los grupos de desarrollo y testeo específicamente autorizados. Los ambientes de desarrollo a menudo son configurados de forma menos segura quelos ambientes de producción y los atacantes pueden utilizar estas diferencias para

descubrir vulnerabilidades o como un camino para poder explotarlas.

- ☐ Implementar un Sistema de Control de Cambios del Software para manejar y registrar los cambios al código tanto para los ambientes de desarrollo como para los de producción.

Seguridad de Base de Datos

- ☐ Utilice *consultas parametrizadas* con tipos de datos fuertemente tipados.
- ☐ Utilice validación de las entradas y codificación de las salidas y asegúrese de manejar los meta caracteres. Si esto falla, no ejecute el comando de la base de datos.
- ☐ Asegúrese de que todas las variables tengan tipos de datos asociados.
- ☐ La aplicación debe de utilizar el mínimo nivel de privilegios cuando accede a la base de datos.
- ☐ Utilice credenciales seguras para acceder a la base de datos.
- ☐ Las cadenas de conexión a la base de datos no deben de estar incluidas en el código de la aplicación. Las cadenas de conexión a la base de datos deben de estar en un archivo de configuración separado en un sistema confiable y debería de estar encriptado.
- ☐ Utilice procedimientos almacenados para abstraer el acceso a los datos y elimine los permisos de las tablas en la base de datos.
- ☐ Cierre la conexión a la base de datos tan pronto como sea posible.
- ☐ Remueva o cambie todas las contraseñas administrativas por defecto. Utilice contraseñas fuertes o frases o implemente autenticación de múltiples factores.
- ☐ Deshabilite todas las funcionalidades innecesarias de la base de datos (por ejemplo: procedimientos almacenados innecesarios, servicios no utilizados, paquetes de utilidades, instale solo el conjunto mínimo de funcionalidades y opciones requeridas (reduce el área de ataque)).
- ☐ Eliminar el contenido innecesario incluido por el proveedor (por ejemplo: esquemas de ejemplo).
- ☐ Deshabilitar las cuentas por omisión que no son necesarias para la operativa del negocio.
- ☐ La aplicación debería conectarse a la base de datos con credenciales diferentes para cada nivel de confianza (por ejemplo: usuarios, usuarios solo lectura, invitados, administrador).

Manejo de Archivos

- ☐ No utilizar directamente información provista por el usuario en ninguna operación dinámica.
- ☐ Exigir autenticación antes de permitir la transferencia de un archivo al servidor.
- ☐ Permita transferir al servidor únicamente los tipos de archivo (por ejemplo: pdf, doc, etc) requeridos por la operativa del negocio. Verificar que la extensión de los ficheros

que utilizamos estén visibles en todo momento.

- ☐ Validar los tipos de archivo transferidos verificando la estructura de los encabezados. La validación del tipo de archivo únicamente por la extensión no es suficiente.
- ☐ No guardar los archivos transferidos en el mismo contexto que la aplicación web. Los archivos deben almacenarse en un repositorio específico o en una base de datos.
- ☐ Evite o restrinja la transferencia de archivos que puedan ser interpretados por el servidor web (por ejemplo: asp, php, jsp, etc).
- ☐ Eliminar los permisos de ejecución a los archivos transferidos.
- ☐ Implemente una transferencia de archivos segura en UNIX mediante el uso de discos lógicos y el uso de las rutas (path) correspondientes o mediante la utilización de un entorno chroot.
- ☐ Cuando se referencie a un archivo existente en el servidor, utilizar una lista blanca de nombres y extensiones válidas. Validar el contenido del parámetro pasado contra esta lista blanca y si se encuentra una coincidencia, denegar la operación o utilizar transferir un archivo pre definido.
- ☐ No utilizar información provista por el usuario para la generación de redirecciones dinámicas. Si se debe proveer esta funcionalidad, la redirección debe aceptar únicamente caminos relativos dentro de la URL previamente establecidos (listablanca de directorios relativos).
- ☐ No incluir en parámetros nombres de directorios o rutas de archivos, en su lugar utilizar índices que internamente se asocien a directorios o rutas pre definidas.
- ☐ Nunca envíe la ruta absoluta de un archivo al cliente.
- ☐ Asegúrese que los archivos y recursos de la aplicación sean de solo lectura.
- ☐ Revise los archivos transferidos por los clientes en busca de virus y malware.

Manejo de Memoria

- ☐ Utilice controles en la entrada y la salida de información no confiable.
- ☐ Revise dos veces que el largo de los buffer sean los requeridos y especificados.
- ☐ Cuando utilice primitivas que requieran el número de bytes a copiar, como ser strncpy(), sea consciente que si el tamaño del buffer destino es igual al tamaño del buffer origen, el destino podría quedar son el NULL-byte requerido del final.
- ☐ Verifique los límites de los buffers si se llama a las funciones dentro de un loop y asegúrese de no correr riesgo de escribir fuera del espacio reservado.
- ☐ Truncar el largo de todos los strings de entrada a un tamaño razonable antes de pasarlos a una función de copia o concatenación.
- ☐ Específicamente libere los recursos, no confíe en el garbage collector (por ejemplo: objetos de conexión, handlers de archivos, etc.)
- ☐ Utilice stacks no ejecutables cuando sea posible (NX bit).
- ☐ Evite el uso de primitivas con vulnerabilidades conocidas (por ejemplo: printf, strcat, strcpy, etc).

- ☐ Libere adecuadamente la memoria previa a la salida de una función y de todos los puntos de finalización de la aplicación.

Practicas Generales para la Codificación

- ☐ Para las tareas habituales, utilice código probado y verificado en lugar de crear códigos específicos.
- ☐ Utilice las APIs previstas para el acceso a funciones específicas del sistema operativo. No permita que la aplicación ejecute comandos directamente en el sistema operativo, menos aún mediante la invocación de una shell.
- ☐ Utilizar funciones de control de integridad (hash, CRC o checksum) para verificar la integridad del código interpretado, bibliotecas, ejecutables y archivos de configuración previo a su utilización.
- ☐ Utilice locks para evitar múltiples accesos simultáneos a los recursos o mecanismos de sincronización (por ejemplo: semáforos) para evitar condiciones de borde (race conditions).
- ☐ Proteja las variables y recursos compartidos de accesos concurrentes inadecuados.
- ☐ Explícitamente inicialice todas las variables y mecanismos de almacenamiento de información (por ejemplo: buffers), durante su declaración o antes de usarlos por primera vez.
- ☐ Las aplicaciones que requieran privilegios especiales, deberán elevar los privilegios solo cuando sea necesario y devolverlos (bajar privilegios) lo antes posible. Mantener los privilegios especiales únicamente cuando sea estrictamente necesario.
- ☐ Evitar errores de cálculo comprendiendo la forma en que el lenguaje de programación maneja las operaciones matemáticas y las representaciones numéricas. Prestar especial atención a las discrepancias en la cantidad de bytes utilizados para la representación, la precisión, diferencias entre valores con y sin signo, truncamiento, conversiones y casting entre tipos de variables, cálculos “no- numéricos” y como el lenguaje maneja los números demasiado grandes o demasiado pequeños para su representación.
- ☐ No utilizar datos provistos por el usuario para ninguna función dinámica.
- ☐ Evite que los usuarios introduzcan o modifiquen código de la aplicación.
- ☐ Revisar todas las aplicaciones secundarias, código provisto por terceros ybibliotecas para determinar la necesidad del negocio para su utilización y validar el funcionamiento seguro, ya que estos pueden introducir nuevas vulnerabilidades.
- ☐ Implementar mecanismos seguros para las actualizaciones. Si la aplicación realiza actualizaciones automáticas, utilizar firmas criptográficas para el código y asegurarse que el cliente que descarga la aplicación verifique dichas firmas. Utilizar canales encriptados para las transferencias de código desde el servidor de actualización.

Glosario

Agente de Amenaza: Cualquier entidad que puede poseer un impacto negativo en el sistema. Puede ser desde un usuario malicioso que desea comprometer los controles de seguridad del sistema; sin embargo, también puede referirse al mal uso accidental del sistema o a una amenaza física como fuego o inundación.

Autenticación: Conjunto de Controles utilizados para verificar la identidad de un usuario o entidad que interactúa con el software

Autenticación Multi-Factor: Proceso de autenticación que le requiere al usuario producir múltiples y distintos tipos de credenciales. Típicamente son basados en algo que el usuario tiene (por ejemplo: una tarjeta inteligente), algo que conoce (por ejemplo:un pin), o algo que es (por ejemplo: datos provenientes de un lector biométrico).

Autenticación secuencial: Cuando los datos de autenticación son solicitados en páginas sucesivas e vez de ser solicitados todos de una sola vez en una única página.

Canonicalizar: Convertir distintas codificaciones y representaciones de datos a una forma estándar predefinida.

Caracteres considerados peligrosos: Cualquier carácter o representación codificada de un carácter que puede afectar la operación intencionada de la aplicación o sistema asociado por ser interpretado de una manera especial, fuera del uso intencionado del carácter. Estos caracteres pueden ser utilizados para:

- Alterar la estructura de las sentencias o código existente
- Insertar código inintencionado
- Alterar caminos
- Causar salidas inesperadas de rutinas o funciones
- Causar condiciones de error
- Obtener cualquiera de los efectos anteriores sobre el flujo de aplicaciones o sistemas

Caso de Abuso: Especificación de un tipo de interacción completa entre un sistema y uno o más actores, donde los resultados de la interacción son perjudiciales para el sistema, uno de los actores o uno de los interesados en el sistema

Codificación de Entidades HTML: Proceso por el cual se reemplazan ciertos caracteres ASCII por sus entidades equivalentes en HTML. Por ejemplo: este proceso reemplazaría el carácter de menor "<" con su equivalente en HTML "<". Las entidades HTML son 'inertes' en la mayoría de los intérpretes, especialmente en los navegadores, pudiendo mitigar ciertos tipos de ataque en los clientes.

Codificación de Salida: Conjunto de controles que apuntan al uso de una codificación para asegurar que los datos producidos por la aplicación son seguros.

Codificación de Salida Contextualizada: Basar la codificación de salida en el uso que le dará la aplicación. El método específico varía dependiendo en la forma que será utilizado.

Confidencialidad: Propiedad de la información por la que se garantiza que está accesible únicamente a entidades autorizadas.

Configuración del sistema: Conjunto de controles que ayuda a asegurar que los componentes de infraestructura que brindan soporte al software fueron desplegados de manera segura.

Consultas parametrizadas (prepared statements): Mantiene la consulta y los datos separados a través del uso de marcadores. La estructura de la consulta es definida utilizando marcadores, la consulta SQL es enviada a la base de datos y preparada, para luego ser combinada con los valores de los parámetros. Esto previene a las consultas de ser alteradas debido a que los valores de los parámetros son combinados con la consulta compilada y con el string de SQL.

Control de Acceso: Un conjunto de controles que permiten o niegan el acceso a un recurso de un usuario o entidad dado.

Control de seguridad: Acción que mitiga una vulnerabilidad potencial y ayuda a asegurar que el software se comporta únicamente de la forma esperada.

Datos de estado: Cuando datos o parámetros son utilizados, ya sea por la aplicación o el servidor, emulando una conexión persiste o realizando el seguimiento del estado de un cliente a través de un proceso multi-pedido o transacción.

Datos del registro de log: Debe incluir lo siguiente:

- Time stamp obtenida de un componente confiable del sistema
- Nivel de severidad para cada evento
- Marcado de eventos relevantes a la seguridad, si se encuentran mezclados con otras entradas de la bitácora
- Identidad de la cuenta/usuario que ha causado el evento
- Dirección IP del origen asociado con el pedido
- Resultado del evento (suceso o falla)
- Descripción del evento

Disponibilidad: Medida de Accesibilidad y Usabilidad del sistema.

Exploit: Forma de tomar ventaja de una vulnerabilidad. Tipicamente se trata de una acción intencional diseñada para comprometer los controles de seguridad del software utilizando una vulnerabilidad.

Falsificación de petición en sitios cruzados (CSRF): Una aplicación externa o sitio web fuerza a un cliente a realizar un pedido a otra aplicación en la que el cliente posee una sesión activa. Las Aplicaciones son vulnerables cuando utilizan parámetros o URLs predecibles o conocidas y cuando el navegador transmite automáticamente toda la información de sesión con cada pedido a la aplicación vulnerable. (Este ataque es discutido específicamente en este documento por ser extremadamente común y poco comprendido).

Frontera de Confianza: Una frontera de confianza típicamente constituye los componentes del sistema bajo control directo. Todas las conexiones y datosprevenientes de sistemas fuera del control directo, incluyendo todos los clientes ysistemas gestionados

por terceros, deben ser considerados no confiables y ser validados en la frontera, antes de permitir cualquier futura interacción con el sistema.

Gestión de Archivos: Conjunto de controles que cubren la interacción entre el código y otro sistema de archivos.

Gestión de memoria: Conjunto de controles de direccionamiento de memoria y uso de buffers.

Gestión de sesión: Conjunto de controles que ayudan a asegurar que la aplicación web maneja las sesiones HTTP de forma segura.

Impacto: Medida del efecto negativo en el negocio que resulta de la ocurrencia de un evento indeseado; pudiendo ser el resultado la explotación de una vulnerabilidad.

Integridad: La seguridad de que la información es precisa, completa y válida, y no ha sido alterada por una acción no autorizada.

Manejo de Errores y Registro en bitácora: Conjunto de prácticas que aseguran que las operaciones de manejo de errores y registro en bitácora se manejan correctamente.

Mitigar: Pasos tomados para reducir la severidad de una vulnerabilidad. Estos pueden incluir remover una vulnerabilidad, hacer una vulnerabilidad más difícil de explotar, o reducir el impacto negativo de una explotación exitosa.

Prácticas Criptográficas: Conjunto de controles que aseguran que las operaciones de criptografía dentro de la aplicación son manejadas de manera segura.

Prácticas de Codificación Generales: Conjunto de controles que cubren las prácticas de codificación que no son parte otras categorías.

Protección de datos: Conjunto de controles que ayudan a asegurar que el software maneja de forma segura el almacenamiento de la información.

Requerimiento de Seguridad: Conjunto de requerimientos funcionales y de diseño que ayudan a asegurar que el software se construye y despliega de forma segura.

Sanitizar: El proceso de hacer seguros datos potencialmente peligrosos a través de la utilización de remoción, reemplazo, codificación o "escaping" de los caracteres que lo componen.

Seguridad de Base de Datos: Conjunto de controles que aseguran la interacción del software con la base de datos de una forma segura y que la base de datos se encuentra configurada de forma segura.

Seguridad de Comunicaciones: Conjunto de controles que ayudan a asegurar que el software maneja de forma segura el envío y la recepción de datos.

Sistema: Término genérico que cubre sistemas operativos, servidores web, frameworks de aplicaciones e infraestructura relacionada.

Validación de entrada: Conjunto de controles que verifican que las propiedades de los datos ingresados coinciden con las esperadas por la aplicación, incluyendo tipos, largos, rangos, conjuntos de caracteres aceptados excluyendo caracteres peligrosos conocidos.
Vulnerabilidad: Debilidad en un sistema que lo hace susceptible a ataque o daño.