Guía introductoria a la Seguridad para el Desarrollo de Aplicaciones WEB

Agosto de 2021

Presentación de la Guía

La programación segura es la práctica de escribir código agotando todos los esfuerzos para que no sea vulnerable. En la actualidad, muchos de los ataques aprovechan las vulnerabilidades presentes en el software. Por eso debe prestarse especial atención al desarrollo seguro de aplicaciones, especialmente aquellas que corren sobre la web y sobre las cuales descansan muchos de los servicios y trámites que los ciudadanos realizan con organismos públicos.

Por lo tanto, es importante incorporar los principios y buenas prácticas de seguridad en todas las etapas del ciclo de vida del desarrollo del software, incluyendo en las etapas iniciales. Esto alcanza también a los nuevos enfoques del desarrollo de aplicaciones, como DevSecOps.

Implementar un esquema de seguridad durante todo el ciclo de vida del desarrollo de software redundará en una menor demanda de recursos, ya que cualquier modificación o cambio debido a vulnerabilidades o fallas de seguridad en etapas posteriores, implicará necesariamente una reingeniería del desarrollo y por tanto más tiempo y mayores costos. En consiguiente, se deben identificar las buenas prácticas y los controles de seguridad en cada una de las etapas e incorporarlas oportunamente.

Atendiendo esta problemática, esta guía fue elaborada con el objetivo de contribuir al desarrollo seguro de aplicaciones en organismos que conforman el Sector Público Nacional. Su texto se basa en un documento redactado por personal técnico[1] perteneciente a lo que es actualmente la DIRECCIÓN NACIONAL DE CIBERSEGURIDAD de la SUBSECRETARÍA DE TECNOLOGÍAS DE LA INFORMACIÓN de la SECRETARÍA DE INNOVACIÓN PÚBLICA DE LA JEFATURA DE GABINETE DE MINISTROS, parte del cual continúa desempeñándose en el área. Se encuentra también publicado en la plataforma GITHUB[2].

Su texto fue enriquecido por los comentarios y opiniones de especialistas en la materia, a quienes se agradece su contribución.

Adicionalmente, como Anexos 1 y 2 se agrega un mayor detalle de OWASP TOP 10 y BSIMM para quienes quieran profundizar sus conocimientos del tema.

[1] El documento original fue elaborado por Rodrigo López Lio, Bruno Bellezza y Francisco Erra.

[2] Disponible en https://github.com/cpeic/Desarrollo-Seguro

Índice

1. Introducción

El software de calidad responde a un proceso de desarrollo seguro. Integrar actividades de seguridad al ciclo de vida de desarrollo de software, si bien puede incidir inicialmente sobre los plazos y el presupuesto de un proyecto, contribuye a una operación con menor porcentaje de fallas y especialmente, a una menor exposición a actividad maliciosa de la información que procesa la aplicación. Tarde o temprano, no hacerlo demandará mayores recursos y tendrá un mayor impacto negativo en los usuarios y en el organismo.

Las aplicaciones seguras asisten en la protección de la propiedad intelectual y la reputación de una organización, ayudan a sostener el funcionamiento de sus procesos y al cumplimiento de la Ley 25.326 de Protección de Datos Personales.

Este documento está dirigido a quienes llevan adelante funciones de desarrollo de software, así como a responsables de áreas de Sistemas, Tecnología y Seguridad de la Información del Sector Público Nacional. Busca difundir buenas prácticas de aplicación a todo el Ciclo de Desarrollo de Software.

2. Modelo Simplificado de Ciclo de Desarrollo

Existen varios modelos de ciclo de vida de desarrollo seguro, cada uno con su enfoque particular (regulatorio, aplicaciones Web, desarrollo para Windows, etc). Se puede seguir alguno de estos modelos, si correspondiere, o se puede tomar el modelo simplificado que sugerimos a continuación. En la sección 3 se ofrece una comparativa de los distintos modelos.

2.1 Inicio del Proyecto

Se identifica la oportunidad de mejorar un proceso y nace la idea de implementar una solución tecnológica basada en un desarrollo interno o tercerizado. Los responsables de seguridad de la información deben intervenir desde el inicio del proyecto, considerando los riesgos involucrados y el posible impacto sobre la información que gestiona el organismo y sobre los procesos involucrados.

Algunas de las actividades de Seguridad a desarrollar en esta etapa son la asignación de recursos, el análisis del contexto, la definición de responsabilidades y la revisión de Marco Normativo aplicable.

2.2 Análisis de Requerimientos.

Se definen las funcionalidades a cubrir con la aplicación sobre la base de los procesos y reglas del organismo y se establecen prioridades.

Es la etapa en la que más se influye sobre el resultado final del proyecto, siendo también indispensable la participación de los responsables de seguridad y privacidad para definir requerimientos específicos. En esta etapa se deben tener en cuenta los requerimientos de cumplimiento establecidos en el marco normativo.

Algunas de las actividades de Seguridad incluyen el análisis de riesgos y establecimiento de los requerimientos de seguridad.

2.3 Diseño del Sistema.

Se determinan los elementos que establecen cómo cumplirá el sistema los requerimientos identificados durante el análisis de requerimientos.

Se podrían incorporar vulnerabilidades al sistema al tomar decisiones incorrectas sobre la arquitectura de diseño del sistema. Las actividades de seguridad durante la etapa de diseño minimizan la necesidad de modificar código en etapas posteriores.

Algunas de las actividades de Seguridad incluyen la definición de los principios de Diseño Seguro, las revisiones de Diseño, y el modelado de Amenazas.

2.4 Etapa de Implementación.

Se escribe el código y se integran componentes construidos previamente en función del diseño establecido. Integrar las actividades de seguridad a esta etapa minimiza la cantidad de errores o fallos a repararse en etapas posteriores.

Algunas de las actividades de Seguridad incluyen Técnicas de programación defensiva, Análisis estático, Revisiones del código.

2.5 Etapa de Pruebas.

Comparando el resultado de la implementación con los requerimientos preestablecidos, pueden detectarse errores o fallas de diseño o codificación, por lo que deben realizarse correcciones y ajustes. Nunca se deben relegar las consideraciones de seguridad para su atención en esta etapa. Las pruebas deben servir como punto de verificación y control adicional para aquellas cuestiones vinculadas a la protección de los datos.

Algunas de las actividades de Seguridad incluyen Pruebas de caja negra y caja blanca y auditorías de código a cargo de terceros.

2.6 Despliegue.

Se prepara la infraestructura sobre la que se ejecutará la aplicación, se instala y configura el software. Una correcta preparación de todo el conjunto de tecnologías puede prevenir vulnerabilidades graves.

Algunas de las actividades de Seguridad incluyen: hardenizado de Sistemas Operativos, así como correcta instalación y configuración de Servicios.

2.7 Mantenimiento.

Se realiza un monitoreo de registros de seguridad y comportamiento de usuarios. Se reciben reportes de incidentes y fallos. Se corrigen errores e implementan actualizaciones.

Algunas de las actividades de Seguridad incluyen Monitoreo de Registros, Respuesta a Incidentes, Desarrollo y Aplicación de Parches.

3. Comparativa: Actividades en Modelos de SDLC Seguro.

Comparativa de conjuntos de actividades de seguridad para las etapas del SDLC sugeridas en documentos de distintas organizaciones.

Etapa/Modelo	OWASP	(ISC)^2 CSSLP	Microsoft SDL	NIST SP800-64	Sugerido
Inicio del Proyecto	Definir alcance Proyecto Definir responsables Comunicar a los participantes de la iniciativa de seguridad.	Políticas de Privacidad. Regulaciones, Privacidad y Cumplimiento.	Entrenamiento de seg. Fundamental.	Iniciar Plan de Seguridad Categorizar Información Evaluar Impacto a la organización. Evaluar Impacto Privacidad. Asegurar uso SDLC Seguro.	Evaluar impacto sobre la organización y la privacidad de datos.
Requerimientos	Evaluar prácticas vigentes. Determinar Madurez (SAMM) Definir objetivos e impacto. Revisión Requerimientos de Seguridad Riesgo usuarios y arquitectura	Fuentes de Requerimientos de Seguridad. Casos de abuso. Matriz RTM.	Requerimientos de Privacidad y Seguridad. Calidad y nivel de falla aceptables. Análisis de Riesgos Privacidad/Seguridad	Evaluar riesgos al sistema. Elegir y documentar controles de seguridad.	Requerimientos de seguridad. Análisis de riesgos.
Diseño	Revisión Diseño Seguro. Revisión seguridad externa. Análisis Riesgos Diseño.	Evaluación de Superficie de Ataque. Modelados de Amenaza.	Requerimientos Diseño Reducción de la superficie de ataque Modelado de Amenazas	Diseñar arquitectura Seguridad Ingeniería Seguridad y Controles Documentación de Seguridad	Principios del diseño Seguro. Modelado de Amenazas.
Implementación	Análisis estático de código Entrenamiento a Desarrollo Estándares de Código	Vulnerabilidades comunes. Programación Defensiva. Procesos seguros.	Herramientas Aprobadas. Deprecar Funciones Inseguras Análisis Estático.	Integrar seguridad a ambientes establecidos.	Prácticas desarrollo seguro. Análisis estático de código. Revisión código entre pares.
Pruebas	Métricas de Seguridad Revisión de Pruebas. Pruebas de Penetración	Pruebas QA Pruebas de Seguridad Gestión Resultado	Análisis Dinámico. Fuzz Testing. Revisión Superficie de Ataque.	Evaluar seguridad del sistema Autorizar sistema de información.	Análisis dinámico de código. Prueba de penetración.
Despliegue	Gestión Riesgos Pre-Deploy Gestión de Infraestructura. Pruebas de Penetración.	Hardenizado Configuración. Publicación Vers.	Plan Respuesta Incidentes Revisión Final de Seguridad Certificación de Versión y archivado	Ver Preparación operativa. Gestión de configuración.	Hardenizado de sistema operativo y servicios.
Mantenimiento	Gestión de respuesta a incidentes. Parches de seguridad. Revisiones de permisos.	Monitoreo. Respuesta a incidentes. Backups y Recuperación	Ejecutar plan de respuesta a incidentes.	Monitoreo continuo	Respuesta a incidentes. Monitoreo de Seguridad Aplicación de parches.

4. Consideraciones al Iniciar un Proyecto

Incorporar un enfoque de seguridad desde el inicio del proyecto reduce los esfuerzos de desarrollo y la cantidad de vulnerabilidades antes de llegar a poner la aplicación operativa o en producción. También ayuda a acortar los plazos para la puesta en producción y aumentar los niveles de seguridad de la aplicación.

4.1 Van a Atacar la aplicación

Se debe operar bajo la premisa de que la aplicación va a recibir ataques variados periódicamente. Dependiendo de su criticidad y de los datos a los que da tratamiento, puede aumentar la intensidad y sofisticación de los ataques.

4.2 Algún ataque va a funcionar

Se debe partir de la premisa de que algunos ataques van a funcionar, ya sea por errores de diseño o fallos en la implementación o por vulnerabilidades en la infraestructura en la que se ejecuta la aplicación.

Siempre debe considerarse la posibilidad real de que un atacante evada los controles de seguridad establecidos. En este contexto, se deben aplicar principios de desarrollo seguro para minimizar el impacto de los ataques exitosos. La postura debe ser que cuando un atacante obtenga acceso al sistema, se debe minimizar el impacto contra la organización y los datos que son objeto de procesamiento electrónico.

4.3 Privacidad de los Usuarios

La información personal de sus usuarios, una vez filtrada no volverá a ser privada. Además, una filtración de datos puede afectar los procesos y la reputación de la organización. Una vez tomada la decisión de procesar y almacenar datos personales debe asumirse la responsabilidad de proteger su confidencialidad e integridad mediante controles de seguridad y técnicas de desarrollo seguro.

4.4 Considerar la seguridad en cada decisión

Cada cambio sobre la aplicación implica nuevos riesgos. Al tomar decisiones sobre la arquitectura del sistema, debe hacerse un análisis de los riesgos implicados. Los riesgos generados con los cambios pueden ser difíciles de detectar. Las decisiones importantes requieren la intervención de los miembros más experimentados del equipo y de los responsables de seguridad.

4.5 Intervención del Personal de seguridad

La participación del equipo de seguridad puede ofrecer una visión especializada sobre los riesgos del desarrollo y ayudar a prevenir fallas, haciendo seguro el diseño de la aplicación. La ventaja de incorporar la visión de seguridad antes de la implementación es que se ahorra el esfuerzo en recursos y tiempo de modificar código para corregir vulnerabilidades.

Los responsables de seguridad deben brindar entrenamiento sobre las buenas prácticas de desarrollo seguro al resto de los miembros del equipo. Todas las partes con algún tipo de responsabilidad en el proyecto de desarrollo pueden beneficiarse recibiendo entrenamiento de seguridad y aprendiendo

los riesgos inherentes a producir nuevas aplicaciones.

5. Durante el Análisis de Requerimientos

A continuación, se listan actividades de seguridad que se pueden integrar a una etapa de análisis de requerimientos.

5.1 Clasificación de Activos.

¿Qué se está defendiendo? La actividad de clasificación de activos consiste en identificar los elementos de la aplicación que ameritan defenderse y estimar su valor para la organización. Ejemplos de activos son: información sensible, componentes de software y servicios.

5.2 Casos de Abuso

Algún atacante intentará vulnerar los controles de seguridad establecidos o la política de uso aceptable. Los casos de abuso enumeran situaciones en las que un atacante intenta vulnerar la seguridad de la aplicación. Esta práctica se realiza en paralelo al estudio de casos de uso.

5.3 Requerimientos de Seguridad

¿Qué no debe hacer la aplicación ante un ataque? Los requerimientos de seguridad definen restricciones sobre la funcionalidad de la aplicación, en base a las reglas de la organización, los activos a defenderse y las posibles amenazas. Se recomienda formular los requerimientos de seguridad en forma explícita, precisa, completa y no conflictiva y usando afirmaciones positivas para validar su cumplimiento.

5.4 Requerimientos de Privacidad

Algunos requerimientos de seguridad refieren específicamente a la confidencialidad sobre elementos de la aplicación, como código fuente o datos privados. Se debe optar por minimizar los datos privados almacenados en sistemas informáticos y justificar explícitamente su almacenamiento.

5.5 Requerimientos Arbitrarios

Refieren al tipo de requerimientos que pueden disminuir la seguridad de la aplicación. Generalmente son incorporados por intervención de personal no especializado. Si bien las aplicaciones desarrolladas deben adaptarse a los procesos de la organización, los expertos deberán decidir sobre la inclusión de requerimientos y su impacto sobre la aplicación a producirse.

5.6 Análisis de Riesgos.

Consiste en estimar la probabilidad de que ocurran ciertos eventos, y evaluar cuál sería el impacto de los mismos para la organización. Permite administrar la asignación de recursos limitados para protegerse de un abanico ilimitado de amenazas posibles. Ofrece planes de acción racionales ante un futuro incierto.

5.7 Priorización de Requerimientos

Es necesario establecer prioridades relativas entre los requerimientos enumerados, para decidir en qué orden se los incorpora al diseño y cuales son descartados.

6. Principios para un Diseño Seguro.

Algunas vulnerabilidades pueden incorporarse a la aplicación a partir de decisiones de diseño incorrectas. Aplicar principios de diseño seguro ayuda a prevenir este tipo de fallos.

6.1 Minimizar la Superficie de Ataque.

Cada línea de código es un bug en potencia. Cada bug puede implicar en una vulnerabilidad y ser explotada. Se debe reducir la cantidad de componentes y librerías de las que se dependerá. Esta práctica afecta la cantidad final de errores en el código.

6.2 Diseñar para ser Mantenido

Indefectiblemente, habrá que corregir errores o fallos. Sin embargo, la arquitectura de la aplicación puede perjudicar o ayudar en su mantenimiento. Será más viable corregir vulnerabilidades de seguridad en un sistema con arquitectura simple y mantenimiento sencillo.

Factores como la complejidad de módulos y arquitectura afectan a la seguridad de la aplicación por el efecto que tienen sobre su mantenibilidad. La rapidez con la que se corrigen los errores afecta la "ventana de vulnerabilidad", siendo ésta el plazo durante el cual se desconoce la existencia de una falla de seguridad en el sistema.

6.3 El eslabón más débil.

Generalmente, los atacantes utilizarán inicialmente las vulnerabilidades más fáciles de explotar. La arquitectura de seguridad de su aplicación será tan resistente como lo sea el componente con la mínima seguridad.

Es una buena práctica identificar los puntos débiles de la seguridad de la aplicación mediante revisiones de diseño y una vez detectados, fortalecerlos ajustando el diseño o implementando controles de seguridad.

6.4 Seguridad por Defecto.

Los usuarios usualmente tienden a no configurar los parámetros de seguridad por lo que este tipo de controles deben ser configurados de la forma más segura posible. Eventualmente se puede ofrecer la posibilidad de deshabilitar algunos de ellos. Como principio general, se recomienda desalentar configuraciones inseguras.

6.5 Mantener la Usabilidad.

El buen software es transparente para el usuario. Es esencial negociar un equilibrio entre controles de seguridad y la usabilidad del sistema. Este principio también se conoce como "Aceptabilidad psicológica". Deberá evitarse generar experiencias de usuario que puedan confundir a los usuarios y llevarlos a tomar malas decisiones de seguridad.

6.6 Autorización para Todo por Defecto.

Por defecto, debe requerirse autorización para acceder a los recursos de la aplicación. En caso de que se decidiera publicar ciertos recursos, debe fundamentarse tal decisión en base a reglas de la organización. Adicionalmente, debería hacerse un análisis de los riesgos asociados a la decisión que se adopte.

6.7 Principio de Mínimo Privilegio

Establece que se debe asignar a un componente o usuario solo los permisos necesarios y suficientes para realizar una acción específica. Estos permisos deben asignarse lo más tardíamente posible es decir, lo más cercano en tiempo al momento en que debe ejecutarse la acción y se deben revocar en cuanto no son necesarios.

6.8 Separación de responsabilidades y roles.

Cuando un componente falle, que no afecte el resto del sistema. Este principio consiste en aislar los privilegios de los componentes para evitar que interfieran entre sí en caso de que alguno sea vulnerable.

6.9 Defensa en profundidad.

Consiste en establecer controles de seguridad consecutivos que seguirán en pie independientemente de que falle alguno de ellos. Este principio de diseño aumenta notablemente la resistencia del sistema, ofrece defensa contra ataques que encadenan múltiples técnicas de explotación.

6.10 Los controles en el Cliente no son suficientes.

Sin control sobre el dispositivo en que se ejecuta el código es inviable controlar su comportamiento. Cualquier control implementado en un equipo que le pertenece a terceros puede ser modificado o deshabilitado. Nunca se debe confiar en datos validados en un equipo cliente, un atacante podría manipularlos arbitrariamente.

6.11 Ayudar a los Administradores.

El software no puede defenderse a sí mismo de atacantes. La aplicación debe ofrecer herramientas útiles a los administradores para analizar el comportamiento de los usuarios y detectar casos de abuso. También debe permitir conceder, analizar y revocar permisos de acceso a usuarios.

Debe ofrecerse un registro de eventos de seguridad, preferiblemente con reducción de falsos positivos y ruido. Nunca mostrar información personal de los usuarios a los administradores. Se debe proteger especialmente interfaces de acceso administrativo, y considerar implementar autenticación multifactor.

6.12 Diseño sin secretos.

La buena seguridad no requiere secretos. Se debe diseñar el sistema bajo la premisa de que eventualmente el público conocerá los detalles de su funcionamiento interno. Seguridad mediante oscuridad es el error de confiar en secretos como controles válidos. Existen técnicas de testeo a ciegas e ingeniería inversa para obtener información sobre el funcionamiento de sistemas teóricamente secretos. No se debe confiar en el secreto de los mecanismos como medida válida de seguridad.

6.13 Modelado de amenazas.

Ofrece herramientas para estudiar y mejorar los diseños propuestos para la aplicación. Ayuda a identificar puntos débiles y fortalece la seguridad total del sistema. Ayuda a prevenir vulnerabilidades a nivel arquitectura de diseño.

- Se estudia el diseño tentativo del sistema, reduciéndolo a sus componentes principales.

- Se grafican la estructura y las relaciones de confianza que existen entre los componentes.
- Se producen diagramas de flujos de datos.
- Se definen posibles amenazas para las relaciones entre componentes.
- Se clasifican las amenazas utilizando criterios STRIDE, DREAD, OCTAVE, etc., en todos los casos priorizando por criticidad.

El modelado de amenazas es una práctica que ofrece un excelente valor agregado sobre el diseño de la aplicación. Puede ayudar a prevenir múltiples tipos de vulnerabilidades graves, producto de relaciones entre componentes. Ayuda a incorporar el principio de defensa en profundidad al diseño de seguridad.

7. Seguridad en procesos de implementación

Las siguientes consideraciones están orientadas a incrementar la seguridad en el proceso de implementación.

7.1 Seguridad de Herramientas

Se deberá consultar la documentación de buenas prácticas de seguridad específicas para los lenguajes de programación, frameworks y librerías a utilizarse.

Considerar las recomendaciones para la herramienta publicadas por el fabricante y otras autoridades afines sobre: Configuraciones de seguridad y reporte de errores, Módulos de seguridad específicos, Patrones de implementación seguros y Existencia de vulnerabilidades conocidas.

7.2 Mantenibilidad y Seguridad

La facilidad para mantener el código incide sobre el tiempo y precisión de la corrección de vulnerabilidades. Además, minimiza significativamente la cantidad de vulnerabilidades que podrían afectar a sistemas productivos.

Una documentación del diseño de arquitectura y las funcionalidades críticas de la aplicación asiste en el mantenimiento del sistema, especialmente en caso de incorporar nuevos miembros al equipo.

Se recomienda establecer guías de estilo y estructura de código. Y es muy preferible documentarlas o utilizar guías ya publicadas por el fabricante de la herramienta.

7.3 Sistemas de Control de Versiones

Herramientas de control de versiones como Git, Subversion, Mercurial y CVS; ofrecen un excelente valor agregado al equipo en materia de organización de código y auditoría de cambios. Con la desventaja de requerir cierto entrenamiento inicial para utilizarse correctamente.

Ayudan a establecer responsabilidades sobre cambios en el código y a generar un feed-back de mejora de las habilidades de programación individuales y del equipo.

7.4 Seguimiento de errores y fallos

Se recomienda establecer un criterio de rango de errores y fallos, en función de su prioridad y la severidad del defecto que generan. También puede estimarse la complejidad para repararlos para ayudar a decidir prioridades.

Los principales sistemas de control de versiones ofrecen módulos o compatibilidad con herramientas de seguimiento de fallos. Siempre se debe verificar una corrección válida y que no se hayan producido nuevos fallos.

7.5 Prudencia al confiar en terceros.

Cada componente puede fallar. Se debe minimizar la confianza y responsabilidad que se deposita en componentes desarrollados por terceros. Se recomienda especial selectividad al incorporar módulos y librerías, revisar las especificaciones y documentación del fabricante.

7.6 Arquitectura de red

Si bien las cuestiones vinculadas a la seguridad en la implementación y configuración de la infraestructura tecnológica en la que se publicará la aplicación web, se encuentran fuera del alcance del presente documento, se consideró importante dada su relevancia, agregar un apartado al respecto.

El diseño de redes seguras es el proceso de diseñarlas considerando las posibles amenazas a las que se verían expuestas y las contramedidas a adoptar. Se debe considerar la implementación de estas medidas sea escalable y sostenible por lo que requerirá un diseño acorde, considerando los riesgos y los principios de seguridad a aplicar.

Se recomienda estandarizar implementaciones, mantener diseños simples pero seguros, identificar eventos a monitorear, documentar su diseño, arquitectura final y configuraciones. Para su diseño se debe considerar aspectos tales como: aislamiento, separación y compartimentación de la red; políticas de acceso; gestión de las identidades; monitoreo de la red y eventos; correlación de eventos y Resiliencia de la red.

En entornos donde se utilice virtualización por hardware o contenedores, se deberá considerar a su vez, las configuraciones de seguridad de dichas implementaciones.

8. Programación Segura

Criterios de seguridad para incrementar la seguridad del código producido. Capítulo basado en el documento
OWASP: Secure Coding Cheat Sheet, publicado por Owasp Foundation.

8.1 Validar Todas las Entradas.

Implementar correctamente un módulo de validación de entradas es un desafío complejo. Se recomienda utilizar librerías de validación de código abierto, reconocidas y con mantenimiento activo para el lenguaje elegido.

Debe ejecutarse una validación sobre todas las entradas que lea la aplicación, en un equipo confiable y configurado en forma segura. Nunca debe confiarse en un control ejecutado en un equipo del usuario. Siempre se debe realizar la validación antes de procesar los datos de entrada.

Es preferible realizar la validación utilizando un criterio de "lista blanca": prohibir todo y permitir deliberadamente casos aceptables. Cuando una validación falla, debe rechazarse los datos de entrada. Es una buena práctica centralizar las rutinas de validación para facilitar su mantenimiento.

```
entrada_1_sucia = parametro_get()
query_base_datos(entrada_1_sucia)
// NUNCA procesar entradas sin antes validarlas.
```

```
entrada_1_sucia = parametro_get()
entrada_1_limpia = validar(entrada_1_sucia)
query_base_datos(entrada_1_limpia)
```

Esta recomendación resuelve los problemas vinculados a ataques de inyección, XSS (Cross Site Scripting).

8.2 Codificar apropiadamente todas las salidas.

En ocasiones se usan entradas controladas externamente para generar contenido dinámico. Se debe controlar estrictamente el formato y los posibles valores de los resultados de módulos hacia componentes externos.

```
nombre_usuario_sucio = parametro_get()
generar_html(nombre_usuario_sucio)
// Permite atacar el navegador de los usuarios.
```

```
nombre_usuario_sucio = parametro_get()
nombre_usuario_limpio = codificar(nombre_usuario_sucio)
generar_html(nombre_usuario_limpio)
```

Esta recomendación resuelve los problemas vinculados a ataques XSS: phishing, robo de sesiones, robo de información privada, distribución de malware.

8.3 Centralizar las Rutinas de Control.

Esta práctica facilita el mantenimiento de los módulos de control. Minimiza las probabilidades de cometer errores en cada instancia del control. Previene la formación de puntos débiles. Facilita la reutilización de código.

```
entrada_1_sucia = parametro_get()
entrada_2_sucia = parametro_post()
entrada_1_sucia = quitar_puntoycoma(entrada_1_sucia)
entrada_1_sucia = quitar _guiones(entrada_1_sucia)
entrada_1_limpia = quitar _arrobas(entrada_1_sucia)
entrada_2_sucia = quitar _comillas(entrada_2_sucia)
entrada_2_limpia = quitar _puntoycoma(entrada_2_sucia)

// Complica la corrección de controles.
```

```
entrada_1_sucia = parametro_get()
entrada_2_sucia = parametro_post()

entrada_1_limpia = validar(entrada_1_sucia, tipo_de_validacion_1)
entrada_2_limpia = validar(entrada_2_sucia, tipo_de_validacion_2)

// Las rutinas centralizadas son más mantenibles.
```

Esta recomendación resuelve los problemas vinculados a un bajo nivel de mantenimiento del Código.

8.4 Autenticación, contraseñas y sesiones

Se recomienda usar librerías o servicios de autenticación estándares y bien comprobados. Se debería requerir autenticación para todas las páginas y recursos, a menos que justifique que sean públicos. Los controles de autenticación deben ejecutarse en el servidor. Se debería requerir autenticación para operaciones críticas sobre una cuenta de usuario, como modificaciones o acceso a información privada y trámites sensibles.

Se recomienda utilizar librerías o frameworks de gestión de contraseñas estándar y bien comprobadas. Las contraseñas nunca deben transmitirse ni almacenarse en texto plano. Se recomienda deshabilitar la opción de autocomplete para formularios de credenciales en el frontend.

Se recomienda utilizar los controles de gestión de sesión del servicio o frameworks, antes de intentar implementar los propios. Siempre ofrecer una funcionalidad de cierre de sesión y un tiempo de terminación automática.

Esta recomendación resuelve los problemas vinculados al bypass de autenticación, robo de sesiones, robo y destrucción de datos personales.

8.5 Control de Accesos.

Se deben unificar los controles de acceso para todo el sitio. La verificación de controles de acceso debe ejecutarse en el servidor. Además, se debe limitar la cantidad de acciones permitidas por usuario, dentro de una ventana de tiempo. La aplicación solo debe tener permisos de acceso a los archivos que resulten estrictamente necesarios.

Esta recomendación resuelve los problemas vinculados al acceso a datos privados por referencia directa a objetos, manipulación de interfaces administrativas.

8.6 Excepciones y Errores Seguros.

Es necesario prever errores y excepciones que puedan ocurrir durante la ejecución del código. En caso de producirse una condición de error, no se debe revelar a usuarios información del funcionamiento interno del sistema.

Se deberá deshabilitar mensajes de debugging y stack-traces del lenguaje, librerías, frameworks y servicios en entornos productivos.

Esta recomendación resuelve los problemas vinculados a la revelación de información sobre la arquitectura interna de la aplicación.

8.7 Carga de Archivos

Esta funcionalidad es de alto riesgo, debe limitarse estrictamente los tipos de archivos que se permite cargar al servidor y correr un análisis de malware sobre su contenido. Deben almacenarse en ambientes aislados o al menos particiones de disco separadas.

Esta recomendación resuelve los problemas vinculados a ataques de ejecución de código, carga de malware y defacements, entre otros.

8.8 Existencia de backdoors administrativas

Las backdoors administrativas son relativamente inofensivas en entornos de desarrollo que no contienen bases de datos productivas. Pero pueden comprometer información privada o la disponibilidad del servicio en caso de llegar a sistemas productivos.

Si son detectadas en sistemas productivos, pueden resultar en robo y modificaciones de datos, perjudicando la reputación de la organización. En el peor de los casos, un atacante puede descubrirlas y utilizarlas. Las auditorías y revisiones de código entre pares ayudan a detectarlas. Debe documentarse su existencia y garantizarse su eliminación de entornos productivos.

```
if( parámetro_get() == “admin1234” )
   conceder_privilegios_administrador()

// Pone en peligro la seguridad de sistemas productivos.
```

Esta recomendación resuelve los problemas vinculados a casos de abuso con privilegios elevados.

8.9 Recomendaciones generales

Se debería estudiar en detalle la documentación de seguridad específica publicada por el fabricante y referentes del lenguaje de programación, librerías, frameworks, servicios y sistema operativo.

Se recomienda utilizar componentes preexistentes ya comprobados en vez de crear código nuevo para tareas comunes. Validar la seguridad de los componentes elegidos mediante revisiones de código y alertas de seguridad publicadas.

Utilizar controles de integridad sobre el código, librerías, ejecutables y archivos de configuración.

9. Ataques comunes: OWASP Top 10

Conocer los tipos de ataques más prevalentes puede ayudar a prevenirlos y aumentar los niveles de seguridad.

Ataque	Descripción	Amenaza	Mitigaciones
Ataques de Inyección	La aplicación envía datos sin validar a un sistema que interpreta instrucciones. Como motores SQL, LDAP, NoSQL, Comandos al Sistema Operativo, XML, XMTP, etc.	Robo y corrupción de datos. Ataques de denegación de servicio. Puede resultar en compromiso total del equipo.	Utilizar Interfaces parametrizadas. Escapar caracteres peligrosos. Filtrar entradas por "listablanca".
Autenticación o Control de Sesión Incorrecto	Comun en módulos de autenticación y sesión desarrollados desde cero. Estos pueden contener múltiples errores de implementación y diseño. (Funciones de Logout, creación de cuentas, cambio contraseñas, olvidó su contraseña, timeouts, recordarme, pregunta secreta).	Ataques sobre cuentas de usuario. Las cuentas con privilegios elevados suelen ser atacadas.	Utilizar un módulo de autenticación y control de sesión fuerte y sin vulnerabilidades conocidas. Revisar detenidamente el diseño y la implementación de estos controles.
XSS: Cross Site Scripting	Ocurre cuando se genera una página incluyendo datos controlados por un atacante que no fueron correctamente filtrados. Los datos pueden ser almacenados en el servidor o reflejados de algún parámetro o Cabeceras HTTP.	Ejecución de scripts arbitrarios en el navegador de los usuarios. Posibilitando robo de sesiones, ataques phishing, modificaciones de contenido, descarga malware y "browser hijackers".	Filtrar los datos correctamente, en base al contexto en que serán presentados. Utilizar librerías de sanitización de inputs reconocidas. Medida adicional (pero insuficiente) es la utilización de Content Security Policy.
Mecanismo de Control de Acceso Incorrecto	Mecanismo de Control de Acceso Incorrecto Las aplicaciones y las APIs muchas veces usan el nombre o código de un objeto para generar páginas. Estos códigos y nombres pueden ser fáciles de adivinar. Las aplicaciones o las APIs no siempre verifican que el usuario esté autorizado para acceder al recurso solicitado.	Comprometer la toda la funcionalidad o datos accesibles. Robo de datos y abuso de servicio.	Verificar permisos de acceso para cada solicitud. Se puede generar códigos de acceso a recurso individuales para cada sesión de usuario.
Mala Configuración de Seguridad	Puede ocurrir en cualquier capa del stack: sistema operativo, plataforma, servicio web, servicio de aplicación, base de datos, frameworks y código personalizado.	Acceso no autorizado a algunos datos y funcionalidades del sistema. Puede resultar en compromiso total del sistema	Proceso de Hardenizado repetible que garantice cierto aseguramiento del ambiente. Los entornos de Desarrollo, QA y producción deben configurarse en forma idéntica (con contraseñas distintas para cada uno). Proceso de seguimiento e instalación de actualizaciones y parches para TODOS los componentes. Arquitectura de la aplicación segura, segregación de funciones efectiva. Verificación automatizada de config.

Ataque	Descripción	Amenaza	Mitigaciones
Filtración de Datos Sensibles	El fallo más común es no encriptar los datos sensibles o encriptarlos incorrectamente usando claves débiles, gestión de claves pobre o algoritmos débiles.	Compromiso de todos los datos desprotegidos.	No almacenar datos sensibles indiscriminadamente. Encriptar todos los datos sensibles almacenados y enviados. Utilizar algoritmos estándar y claves fuertes. Las contraseñas deben almacenarse utilizando algoritmos diseñados específicamente para proteger contraseñas, (bcrypt, PBKDF2, scrypt). Deshabilitar la opción caché y de autocomplete, en páginas y formularios con información sensible
Falta de Protección contra Ataques	Las aplicaciones y APIs son atacadas todo el tiempo, generalmente detectan los inputs inválidos pero solo los rechazan: permitiendo que los atacantes prueben una y otra vez.	La mayoría de los ataques comienzan con un sondeo de vulnerabilidades. Permitir este tipo de pruebas incrementa las probabilidades de un ataque exitoso, ayuda a los atacantes.	Medidas de detección y respuesta a ataques. Proceso de aplicación de parches críticos (o virtuales).
CSRF: Cross Site Request Forgery	Aprovechando el hecho de que los navegadores envían las cookies de sesión automáticamente, un atacante puede generar un link que solicite una acción sobre la aplicación.	Los usuarios pueden ser engañados para ejecutar acciones no deseadas en la aplicación, como cambios de información, modificaciones de cuentas o compras.	Utilizar tokens únicos para confirmar la validez de una acción. Como medida adicional se puede usar el flag “SameSite=strict” sobre todas las cookies.
Componente con Vulnerabilidades Conocidas	Ocurren cuando no hay aseguramiento de que se usen componentes y librerías actualizadas. O cuando hay dependencias desconocidas entre componentes.	Puede facilitar todo tipo de ataques: de inyección, XSS, controles de acceso rotos, etc. Puede resultar en compromiso total del host y robo de datos.	Inventariar continuamente las versiones de los componentes utilizados. Monitorear bases de datos de vulnerabilidades en busca de los componentes utilizados. Analizar componentes instalados y evaluar si son necesarios. Aplicar actualizaciones y parches de seguridad provistos por el fabricante del componente.
API Desprotegida	Muchas aplicaciones modernas se conectan a APIs en el backend, (XML, JSON, RPC o customizadas) que son susceptibles a todo tipo de ataques. Este tipo de vulnerabilidades, generalmente requiere revisiones manuales y suele permanecer sin ser detectado.	Todo tipo de ataques: robo, corrupción e incluso modificación de datos. Acceso no autorizado a funciones de la aplicación. Puede resultar en compromiso total del equipo.	Asegurar las comunicaciones entre cliente de la aplicación y la API. Esquema de autenticación fuerte. Parser de datos Hardenizado contra ataques. Esquemas de control de accesos. Protección contra ataques de inyección.

10. Pruebas de Seguridad

Criterios para verificar si el diseño y la implementación de la aplicación cumplen con los requerimientos de seguridad establecidos.

10.1 Comienzo Temprano de la Etapa de Pruebas

Cuanto más temprano se descubran los fallos y errores, más económico será corregirlos, y menor será el impacto sobre la integridad del resto del código. Idealmente, debería comenzarse con las pruebas de seguridad en paralelo a la etapa de desarrollo. A efectos de detectar y corregir tempranamente posibles vulnerabilidades.

10.2 Revisiones de Código entre Pares.

Se realiza en paralelo con las tareas de desarrollo. No se requiere una cobertura total del código. Se recomienda comenzar por los componentes más críticos de la aplicación.

Esta práctica reduce drásticamente la cantidad de errores y fallos que permanecen en el código sin ser detectadas. Aumenta los niveles de comunicación entre miembros del equipo, ayuda a consolidar criterios de buenas prácticas e incrementa las habilidades programáticas individuales.

Es preferible establecer una metodología estándar de corrección y validación de errores para todo el equipo. Se recomienda limitar los bloques de revisión a alrededor de 400 líneas de código y períodos de una hora.

10.3 Herramientas de escaneo estático.

El análisis estático de un IDE, compilador, intérprete o herramienta especializada puede ofrecer una excelente medida para prevenir errores o fallos. Este tipo de herramientas complementa a las revisiones manuales para detectar errores sintácticos o que requieren seguir flujos de ejecución muy largos para ser interpretado por humanos.

Las herramientas tienen la debilidad de ofrecer falsos positivos y falsos negativos. Es posible reconfigurar o ajustar las alertas para prevenir resultados falsos. Esta práctica no sustituye las revisiones manuales de código.

10.4 Pruebas de Penetración

Se evalúa la seguridad del sistema desde la perspectiva de un atacante. Cumple la función de detectar vulnerabilidades que escaparon a todos los otros controles de seguridad anteriores.

Por si sola, este tipo de pruebas no es suficiente para garantizar la seguridad de una aplicación. Corregir fallos detectados mediante pruebas de penetración, suele ser mucho menos económico que prevenirlos en etapas tempranas del ciclo de desarrollo.

10.5 Auditorías Manuales de Código

Algunos fallos son difíciles de detectar con herramientas o pruebas de penetración, pero resultan evidentes al leer el código responsable. Este tipo de pruebas suele llamarse White-box o transparent-box.

10.6 Pruebas Tercerizadas y Confidencialidad

En caso de que las pruebas sean realizadas por personal ajeno a la organización se recomienda documentar las condiciones de las actividades de prueba, definiendo: alcance, objetivos, tipos de pruebas y horarios

permitidos.

11. Puesta en producción

Partiendo de una correcta verificación durante la etapa de pruebas, se presentan a continuación buenas prácticas para el despliegue de la aplicación.

11.1 Segregación de ambientes

Se deberá tomar medidas para garantizar una estricta segregación entre los ambientes de desarrollo, pruebas y producción. No se debe conceder acceso irrestricto a entornos productivos. No debe utilizarse datos productivos en entornos de desarrollo y pruebas, se recomienda generar datos ficticios o aleatorios con estructura equivalente a la productiva. Se debe formalizar una serie de controles previo a la puesta en producción de nuevas versiones de la aplicación. Al producirse la migración de código hacia entornos productivos, prestar especial cuidado a la eliminación de todas las "backdoors" administrativas.

11.2 Hardenizado de equipos

Se deben eliminar componentes innecesarios, configurar correctamente los necesarios, activar componentes de seguridad y documentar las configuraciones establecidas para cada componente.

Siempre se debe cambiar las contraseñas preestablecidas por defecto, cumpliendo con la política de contraseñas de la organización.

Deberá prestarse especial atención a una correcta configuración del sistema operativo y los servicios utilizados por la aplicación siguiendo las buenas prácticas de seguridad recomendadas por cada fabricante y comunidad de usuarios expertos.

Se debe separar equipos de servicios web y bases de datos, ubicando el servicio web en la DMZ y el servicio de base de datos en un entorno privado. Se recomienda cifrar las comunicaciones entre ambos servicios.

Las herramientas de virtualización y contenedores ofrecen una capa extra de seguridad generalmente deseable. Sin embargo, requieren mantenimiento adicional y pueden impactar sobre el rendimiento del sistema.

Se recomienda un esquema de particiones que separe el sistema operativo, servicios, los registros de auditoría y archivos cargados por usuarios.

Las cuentas de servicio y administrativas deberán contar con los mínimos privilegios necesarios para realizar las acciones previstas.

Prestar especial consideración a una correcta configuración del servicio de cifrado para datos en tránsito y el cifrado de las unidades de almacenamiento protegiendo: código fuente, bases de datos y datos personales.

Debería correrse pruebas y revisiones de la configuración para todos los componentes. Es preferible lograr una configuración base segura que pueda adaptarse para cada caso de uso.

Se recomienda el uso de herramientas de seguridad específicas para cada capa, que deben seleccionarse cuidadosamente. Red: balanceadores de carga, firewalls, sistemas de detección y prevención de intrusos. Sistema operativo: firewall de host, controles de integridad de archivos, cifrado, scripts de hardenizado, antimalware. Servicios: firewall de aplicaciones, módulos de seguridad, scripts de hardenizado. Aplicación: Correcta configuración, componentes de seguridad, generación registros de auditoría.

12. Mantenimiento

Recomendaciones para mantener los niveles de seguridad durante el funcionamiento productivo de la aplicación.

12.1 Protocolo de Backups

Debe implementarse un protocolo de back-up periódico formalizado, con procedimientos para restaurar servicio en caso de fallos, a efectos de garantizar integridad y disponibilidad. Se recomienda definir tipo de backup, periodicidad, pruebas, usuarios y responsables.

12.2 Monitoreos periódicos de seguridad y alertas

Se deberá ofrecer a los administradores herramientas de administración de cuentas, alertas de seguridad, monitoreo de abuso y registros de auditoría.

12.3 Reporte de incidentes y vulnerabilidades

Se recomienda ofrecer un canal de contacto para el reporte de fallos, errores y vulnerabilidades. Se deberá implementar una Base de Conocimientos para facilitar el seguimiento de cada caso.

Será necesario suscribirse a las notificaciones de seguridad del fabricante de cada componente utilizado en la aplicación. En caso de recibir información de terceros sobre vulnerabilidades que afecten a nuestra aplicación, se recomienda recabar la información necesaria para solucionar lo reportado.

12.4 Ventana de vulnerabilidad

Es el plazo de tiempo desde que se toma conocimiento de la existencia de una vulnerabilidad hasta que se produce una nueva versión o parche correctivo. Se debe minimizarla para reducir el riesgo de ataques. Se recomienda establecer procesos especiales para casos que requieran corrección urgente. Algunos dispositivos permiten mitigar temporalmente las vulnerabilidades mediante la aplicación de “parches virtuales”.

12.5 Actualizaciones de seguridad

Se deberá verificar que las actualizaciones reparen efectivamente la vulnerabilidad o fallo. También deberá probarse que los cambios no generan nuevas vulnerabilidades. En caso de que las actualizaciones se instalen manualmente deberá corroborarse que no queden instancias con versiones vulnerables. Es una buena práctica publicar reportes describiendo información técnica y la criticidad de la actualización. Dependiendo del tipo de aplicación, puede ser necesario notificar a los usuarios de los cambios aplicados.

12.6 Descarte de la aplicación

Al llegar al final de la vida útil de un sistema, por obsolescencia o reemplazo, debe considerarse especialmente la privacidad de los datos almacenados. En caso de migración, deben tomarse medidas para evitar que se comprometa la integridad y establecerse mecanismos de custodia para garantizar la confidencialidad.

Si se necesita destruir los datos de la aplicación original, debe validarse la eliminación mediante sobreescritura y adicionalmente destrucción física de los medios de almacenamiento que fueron utilizados.

Al momento de una migración, deben validarse y sanitizarse todos los datos importados.

13. Referencias

- Acrosec DMZ
  https://www.acrosec.jp/dmz-intro/?lang=en
  https://www.acrosec.jp/dedicated-dmz/?lang=en
- Apple: Secure Coding Guide
  https://developer.apple.com/library/content/documentation/Security/Conceptual/SecureCodingGuide/Introduction.html
- Best Practices for Code Review
  https://smartbear.com/learn/code-review/best-practices-for-peer-code-review/
- Berkeley: Secure Coding Practice Guidelines
  https://security.berkeley.edu/secure-coding-practice-guidelines
- CMU: Secure SDLC Processes
  https://www.sei.cmu.edu/reports/05tn024.pdf
- CWE/SANS: Top 25 Most Dangerous Software Errors
  http://cwe.mitre.org/top25/
- Cybrary: Sunny Wear, Secure Coding
  https://www.cybrary.it/course/secure-coding/
- David Wheeler: Secure Programming HOWTO
  https://www.dwheeler.com/secure-programs/Secure-Programs-HOWTO/
- Diego Spitia Cristian Borghello: Metodologías de Desarrollo Seguro
  https://www.youtube.com/watch?v=eMs2fErek_c
- Harvard: Secure Software Development
  https://canvas.harvard.edu/courses/26673/assignments/syllabus
- ISASecure: Security Development Lifecycle Assessment v3.0
  https://www.isasecure.org/en-US/Certification/IEC-62443-SDLA-Certification-(1)
- Jim Manico: dotSecurity 2017, Secure SDLC
  https://www.youtube.com/watch?v=M7qMP3C5bkU
- López Lio, R. Belleza, B. Erra, F. 2017, Introducción a la Seguridad para el Desarrollo de Aplicaciones
  https://github.com/cpeic/Desarrollo-Seguro
- Maurice Dawson: Secure Software Development Life Cycle
  https://es.slideshare.net/drdawson/secure-software-development-life-cycle
- Microsoft: The Security Development Lifecycle
  https://www.microsoft.com/en-us/securityengineering/sdl
- Mozilla: Web Security
  https://developer.mozilla.org/en-US/docs/Web/Security
- NIST: SP800-27 Engineering Principles for IT Security

http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-27ra.pdf

- NIST: SP800-64 Security Considerations in the SDLC
  http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-64r2.pdf
- OWASP: Guía Para Construir Aplicaciones y Servicios Seguros
  https://www.scribd.com/document/112624664/OWASP-Development-Guide-2-0-1- S panish-pdf
- OWASP: Lista de Verificación de Prácticas de Codificación Segura
  https://owasp.org/www-pdf-archive/OWASP_SCP_Quick_Reference_Guide_SPA.pdf
- OWASP: SAMM 1.0
  https://opensamm.org/downloads/SAMM-1.0.pdf
- OWASP: Security Principles Project
  https://www.owasp.org/index.php/OWASP_Security_Principles_Project
- OWASP: Top 10 Project
  https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project
- Pluralsight: CSSLP Secure Software Concepts
  https://app.pluralsight.com/library/courses/csslp-secure-software-concepts/
- SAFECODE: Fundamental Practices for Secure Software Development
  https://safecode.org/publication/SAFECode_Dev_Practices1108.pdf
- SAFECODE: Software Assuranse, Overview of Current Industry Best Practices
  https://safecode.org/publication/SAFECode_BestPractices0208.pdf
- SAFECODE: Tactical Threat Modeling
  https://safecode.org/wp-content/uploads/2017/05/SAFECode_TM_Whitepaper.pdf
- SCIPP: Secure Web Application Development Awaraness
  https://www.scippinternational.org/wp-content/uploads/2016/03/scipp-swada-gap-new.pdf
- SEI CERT: Top 10 Secure Coding Practices
  https://wiki.sei.cmu.edu/confluence/display/seccode
- SEI CMU: Secure SDLC Processes
  https://resources.sei.cmu.edu/asset_files/WhitePaper/2013_019_001_297287.pdf
- SEI CMU: Secure Coding Best Practices
  https://www.youtube.com/watch?v=cU4dKMo13dk
- Synopsys: BSIMM 7
  https://www.synopsys.com/content/dam/synopsys/bsimm/reports/BSIMM7.pdf
- University of Texas: Minimum Security Standards for Application Development and Administration
  https://security.utexas.edu/policies/standards_application
- US-CERT: Build Security In
  https://us-cert.cisa.gov/bsi

Anexo 1 – Introducción a OWASP Top Ten

El OWASP Top 10 es un documento de concientización para desarrolladores y seguridad de aplicaciones web. Representa un amplio consenso sobre los riesgos de seguridad más críticos para las aplicaciones web.
Este documento contiene 10 categorías cada una contiene 5 secciones:

1. Riesgo
    a. vectores de ataque
    b. debilidades de seguridad
    c. impacto tecnológico y del negocio.
2. ¿La aplicación es vulnerable?
3. cómo se previene
4. Ejemplos de escenarios de ataques
5. Referencias.

Modo de uso

En sí, cada categoría representa un riesgo de seguridad, pero cada riesgo es genérico. Por ejemplo A1: injection representa cualquier tipo de inyección a una aplicación pero existen muchísimas subcategorías de este riesgo (SQL injection, Command injection, Template injection, etc) que deben estudiarse específicamente para obtener resultados. Para estos casos deben usarse las referencias de la categoría que suelen ser muy útiles a la hora de estudiar el riesgo de la categoría de manera más específica. Comúnmente en esta subcategoría podemos encontrar:

- OWASP Proactive Controls
    - Una lista de técnicas de seguridad que deben tenerse en cuenta para cada proyecto de desarrollo de software
    - Están ordenados por orden de importancia, siendo el control número 1 el más importante.
    - Fue escrito por desarrolladores para ayudar a nuevos desarrolladores a asegurar la seguridad en el desarrollo de software.
    - En en OWASP Top Ten hace referencia a un control específico de la categoría.
- OWASP Cheat Sheet:
    - Una colección concisa de información de alto valor sobre temas específicos de seguridad de aplicaciones.
    - En en OWASP Top Ten hace referencia a cheatsheet sobre temas de seguridad que amplían a la categoría.
- OWASP ASVS
    - En en OWASP Top Ten hace referencia a requerimiento específico de ASVS relacionado con la categoría.
- OWASP Testing Guide
    - En en OWASP Top Ten hace referencia al escenario específico de la WSTG relacionado con la categoría.
- OWASP Automated Threats
    - Enumeración de varios ataques automáticos a aplicaciones web

    - También define un lenguaje estándar para referenciar a estos ataques
    - En en OWASP Top Ten hace referencia a un ataque automático específico que está relacionado con la categoría.
- OWASP Vulnerabilities
    - enumeración de vulnerabilidades específicas
    - En en OWASP Top Ten hace referencia a una vulnerabilidad relacionada con la categoría.
- Documentos externos
    - MITRE: CWE/CVE
    - NIST

ASVS (Application Security Verification Standard)

OWASP ASVS ofrece una lista completa de requisitos, controles y pruebas de seguridad de aplicaciones web que puede utilizar para determinar el alcance, crear y verificar aplicaciones web y móviles seguras.
Permite a las organizaciones desarrollar y mantener aplicaciones más seguras; y también brinda a los proveedores de servicios de seguridad y proveedores un conjunto de controles bien documentados con el que pueden alinear sus requisitos y ofertas.
Tiene como objetivo fijar un estándar para normalizar el nivel y el rigor con el que se hacen las verificaciones de seguridad sobre aplicaciones webs.

Modo de uso

- La idea es que:
    - Se use como métrica para dar un valor de confianza a las aplicaciones
    - Se use como guía para los desarrolladores puedan construir controles de manera de satisfacer los requisitos de seguridad de la aplicación
    - Se use durante adquisiciones proveyendo una base para especificar requerimientos de seguridad en los contratos

El estándar para la verificación de seguridad en aplicaciones define tres niveles de verificación para la seguridad, con cada nivel incrementando su profundidad.

1. ASVS Nivel 1 es para todo el software.
2. ASVS Nivel 2 es para las aplicaciones conteniendo datos sensibles, los cuales requieren protección.
3. ASVS Nivel 3 es para las aplicaciones más críticas, aplicaciones realizando transacciones de alto valor, conteniendo datos médicos sensibles, o cualquier aplicación el cual requiere un alto nivel de confianza.

Cada nivel del ASVS contiene una lista de requerimientos para la seguridad.
Cada uno de estos requerimientos puede también ser mapeado hacia características y capacidades específicas en seguridad, los cuales deben ser construidos en el software por los desarrolladores.

WSTG (Web Security Testing Guide)

La WSTG (Web Security Testing Guide) nace en el año 2013 como una guía complementaria al OWASP TOP 10. Este documento provee información más específica y técnica sobre los controles de seguridad que se deben contemplar a la hora de realizar pentesting en aplicaciones web. Además, a la WSTG se le suma un complemento llamado OWASP

Web Application Penetration Checklist que nos ayudará a contabilizar y categorizar los controles de seguridad que debemos ejecutar según OWASP TOP 10 y la información de la WSTG.

El objetivo del proyecto es ayudar a comprender el qué, por qué , cuándo , dónde y cómo testear aplicaciones web. El proyecto ha proporcionado un marco de prueba completo, no una simple lista de verificación o prescripción de problemas que deben abordarse. Los lectores pueden utilizar este marco como plantilla para crear sus propios programas de testing. La WSTG describe en detalle tanto el marco de prueba general como las técnicas necesarias para implementar el marco en la práctica.

El método de testing seguridad de aplicaciones web de OWASP se basa en el enfoque de caja negra. El tester no sabe nada o tiene muy poca información sobre la aplicación que se va a testear. Es muy útil para enfocar un pentest de manera organizada o utilizarlo en pruebas de caja blanca cómo guía para testear de manera organizada.

Modo de uso

La WSTG está dividida en categorías y pruebas. Un escenario es un conjunto de categoría y prueba. Cada escenario tiene un identificador en el formato WSTG-<category>-<number>, donde: category es una mayúscula cadena de 4 caracteres que identifica el tipo de prueba o vulnerabilidad, y number es un valor numérico de 01 a 99. Por ejemplo: WSTG-INFO-02 es la segunda prueba de recopilación de información.

El capítulo 4 es el capítulo más técnico de esta guía. Está dividido en 12 subcapítulos, cada uno detallando una categoría de testing.

1. **Information Gathering:** categoría dedicada a la recopilación de información de una aplicación ya sea activa o pasivamente. En esta categoría se utilizan técnicas cómo fingerprinting, enumeración, identificación y mapeo.
2. **Configuration and Deployment Management Testing**: categoría dedicada al testing de distintas configuraciones de la aplicación, cómo testeo de la infraestructura de la red, plataforma de la aplicación, métodos HTTP/S, permisos de archivos,etc.
3. **Identity Management Testing:** categoría dedicada a testear el manejo de identidad de la aplicación, cómo el registro de usuarios, manejo de roles y cuentas y debilidades de estos procesos.
4. **Authentication Testing:** categoría dedicada a testear la autenticación de la aplicación, cómo maneja las credenciales de los usuarios, testeando si los procesos relacionados con la autenticación son seguros y no permiten password débiles o su transporte por medio no cifrados.
5. **Authorization Testing:** categoría dedicada a testear la autorización de la aplicación testeando si existen maneras de bypasear permisos o escalar privilegios en la aplicación.
6. **Session Management Testing:** categoría dedicada a testear el manejo de sesiones testeado el uso de cookies, si existe manera de vulnerar las sesiones fijándolas, robandolas o adivinándolas.
7. **Input Validation Testing:** categoría dedicada a testear todas las entradas de datos de los usuarios, esta categoría testea cualquier tipo de inyecciones (OWASP A1), también XSS (OWASP A3), HTTP Smuggling, HTTP Pollution.
8. **Error Handling:** categoría dedicada a testear el manejo inadecuado de errores que muestran información sensible y es comúnmente utilizada por los atacantes utilizando fingerprinting.

9. **Weak Cryptography:** categoría dedicada a testear el cifrado de información ya sea en la transmisión de datos (canales de información seguros) cómo en los datos en reposo (almacenamiento criptográfico seguro).
10. **Business Logic Testing:** categoría dedicada a testear la lógica de negocio de la aplicación analizando flujos de trabajo, límites de funciones y tiempos de procesos.
11. **Client-side Testing:** categoría dedicada a testear el frontend de la aplicación testeando XSS, inyecciones HTML o CSS, Clickjacking, test de websockets.
12. **API Testing:** categoría dedicada a testear el uso de APIs

Anexo 2 - Introducción a BSIMM

Introducción

El BSIMM (Building Security In Maturity Model) es un modelo de madurez que sirve para orientar a una organización que desarrolla software con respecto a las acciones que puede encarar con el fin de hacerlo más seguro. En este documento hacemos una introducción somera del modelo y de las formas en que se utiliza.

Modelos de madurez

Los modelos de madurez son herramientas para evaluar objetivamente los procesos propios con respecto a algún criterio en particular. En algunos casos, existen certificaciones y son requeridas (por ejemplo CMMI). Califican las iniciativas por niveles de madurez, en una o más dimensiones. Permiten abordar la evaluación de los procesos propios y dan información sobre iniciativas que permitirían mejorarlos.

Iniciativas de seguridad de software

Llamamos “iniciativa de seguridad de software” a todas las actividades llevadas a cabo con el propósito de construir software seguro. Es un programa aplicado a toda la organización, cuyo objetivo es inculcar, medir, gestionar y hacer evolucionar actividades de seguridad del
software en forma coordinada. También es conocido como “programa empresarial de seguridad del software”

El modelo BSIMM

BSIMM es un modelo de madurez de iniciativas de seguridad de software, basado en relevar las iniciativas de una cantidad de empresas. Es un modelo descriptivo, más que prescriptivo, y por lo tanto normalmente no se certifica; se utiliza como ayuda para guiar la iniciativa de seguridad de software. Al mismo tiempo, proporciona un panorama de qué están haciendo al respecto los diferentes actores.

Modelo de referencia

BSIMM presenta un modelo de referencia, en el que las actividades de una iniciativa de seguridad de software se dividen en cuatro grandes dominios: Gobernanza, Inteligencia, Puntos de contacto con el SSDL, y Despliegue.

- Gobernanza:
    - Abarca la planificación, asignación de roles y responsabilidades, identificación de metas de seguridad del software, determinación de presupuestos e identificación de métricas y puertas de control.
    - Se centra en la identificación de controles para el cumplimiento de marcos regulatorios (PCI DSS, BCRA, HIPAA); el desarrollo de controles contractuales, tanto internos como con terceras partes; la definición de la política de seguridad del software de la organización y su auditoría.
    - Desarrollo de planes de capacitación básica en seguridad del software y orientada a los diferentes roles del proceso de desarrollo de software y tecnología.
- Inteligencia

- Abarca la captura de información que es utilizada para pensar como un atacante: modelado de amenazas, desarrollo de casos de abuso, clasificación de datos y patrones de ataque específicos de tecnología.
- Se centra en la creación de patrones de seguridad, construcción de marcos de referencia de middleware, y creación y publicación de directrices de seguridad proactiva.
- Consiste en la obtención de los requisitos de seguridad de la organización, determinación de qué paquetes de software comercial enlatado recomendar, creación de normas para controles de seguridad (tales como autenticación, validación de entrada) y tecnologías en uso, y la creación de un comité de revisión de normas.

- Puntos de Contacto con el SSDL
  - Abarca la captura de la arquitectura del software en diagramas concisos y la adopción de un proceso de análisis de riesgo de la misma.
  - Incluye el uso de herramientas de comprobación de código, desarrollo de reglas a medida, definición de perfiles por roles para el uso de aplicaciones, análisis manual, y resultados de seguimiento/medición.
  - Se refiere a las verificaciones previas al lanzamiento, incluida la integración de la seguridad dentro de los procesos de aseguramiento de calidad. Incluye el uso de herramientas de seguridad de caja negra, las pruebas de caja blanca guiadas por riesgos, la aplicación de un modelo de ataque y el análisis de cobertura de código.
- Despliegue
  - Implica la realización de pruebas de penetración y la retroalimentación con los resultados de las mismas a la gestión de incidentes de seguridad.
  - Se ocupa de la aplicación de parches del sistema operativo y la plataforma, los firewalls de aplicación en el ámbito web, la documentación de instalación y configuración, la monitorización de aplicaciones, la gestión de cambios y la firma del código.
  - Tiene a cargo el control de versiones, el seguimiento de defectos y las respectivas correcciones.

Dentro de cada dominio hay tres prácticas, y todas las actividades en la iniciativa de seguridad de software se enmarcan en alguna de estas prácticas. El modelo a nivel dominios y prácticas se ve de esta forma:

Dentro de cada práctica, a cada actividad se le asigna un nivel de madurez, y además se reporta qué actividad es la más comúnmente implementada; el nivel de madurez más alto de las actividades de cada práctica será entonces el nivel de madurez de la iniciativa para esa práctica.

Para qué sirve

El modelo BSIMM da una idea objetiva de la madurez de las iniciativas en el área de seguridad del software:

- Como promedios en el mundo
- Como promedios según área de actividad (si hay suficientes muestras)

También permite evaluar la madurez de la propia iniciativa de seguridad del software, respecto de buenas prácticas relevadas en distintos sectores de la industria del software, y finalmente ayuda en la búsqueda de nuevas iniciativas tendientes a mejorar el estado de la seguridad del software.

Cómo se usa

- Como guía para el relevamiento de iniciativas de seguridad del software en la empresa (entrevistas presenciales)
- Como modelo para el reporte de madurez de iniciativas
- Como orientador para la formulación de recomendaciones para los próximos pasos

Al ser un modelo descriptivo, el objetivo es sólo observar e informar; asimismo, se actualiza periódicamente: su versión más reciente se puede encontrar siempre en www.bsimm.com, y una traducción al castellano de BSIMM V se encuentra en http://www.fundacionsadosky.org.ar/wp-content/uploads/2014/07/BSIMM-V-esp.pdf

República Argentina - Poder Ejecutivo Nacional
2021 - Año de Homenaje al Premio Nobel de Medicina Dr. César Milstein

Hoja Adicional de Firmas
Informe gráfico

Número: IF-2021-76718462-APN-DNCIB#JGM

CIUDAD DE BUENOS AIRES
Viernes 20 de Agosto de 2021

Referencia: Anexo I : Guia Introductoria a la Seguridad para el Desarrollo de Aplicaciones WEB

El documento fue importado por el sistema GEDO con un total de 35 pagina/s.

Digitally signed by Gestion Documental Electronica
Date: 2021.08.20 12:47:02 -03:00

Gustavo Raúl SAIN
Director Nacional
Dirección Nacional de Ciberseguridad
Jefatura de Gabinete de Ministros

Digitally signed by Gestion Documental Electronica
Date: 2021.08.20 12:47:04 -03:00