Ministerio de Modernización
Presidencia de la Nación

Introducción a la Seguridad para el Desarrollo de Aplicaciones

Coordinación de Proyectos e Investigación de Ciberseguridad
Dirección Nacional de Infraestructuras Críticas de Información y Ciberseguridad

PAIS DIGITAL

Versión 1
Diciembre de 2017

Contenidos

Equipo de Trabajo

- López Lio Rodrigo

- Bellezza Bruno

- Erra Francisco

1. Introducción

El buen software es seguro. Integrar actividades de seguridad al ciclo de vida de desarrollo de software incide directamente sobre los plazos de desarrollo y el presupuesto de un proyecto.

Las aplicaciones seguras asisten en la protección de la propiedad intelectual y la reputación de una organización. Ayudan a sostener el funcionamiento de procesos de la organización y a garantizar el cumplimiento de la *Ley 25.326 de Protección de Datos Personales*.

Este documento está dirigido a los **equipos de desarrollo de software en Argentina**, busca difundir buenas prácticas a aplicarse durante todo el Ciclo de Desarrollo de Software.

2. Modelo Simplificado de Ciclo de Desarrollo

A continuación se presenta un modelo simplificado de ciclo de desarrollo. Se describe el ciclo independientemente de la metodología preferida por su equipo y de la duración o alcance de las etapas.

2.1. Inicio del Proyecto

Se identifica la oportunidad de mejorar un proceso y nace la idea de implementar una solución tecnológica. Idealmente, los responsables de seguridad deberían intervenir desde el inicio del proyecto. Considerando el posible impacto sobre la seguridad de la organización y sus procesos.

Actividades de Seguridad: *Entrenamiento de Seguridad, Definir Responsabilidades, Revisión de Marco Normativo.*

2.2. Análisis de Requerimientos.

Define funcionalidades sobre de los procesos y reglas de la organización, se establecen prioridades.

La etapa en la que más se influye sobre el resultado final del proyecto, es indispensable la participación de los responsables de seguridad para definir requerimientos específicos. Los requerimientos de privacidad y seguridad deberán basarse tanto en las reglas de la organización y el marco normativo.

Actividades de Seguridad: *Análisis de casos de abuso, Análisis de riesgos, Requerimientos de seguridad.*

2.3. Diseño del Sistema.

Establece las características necesarias para cubrir los requerimientos establecidos. Es posible incorporar **vulnerabilidades al sistema tomando decisiones incorrectas sobre la arquitectura de diseño**. Las actividades de seguridad durante la etapa de diseño minimizan la necesidad de modificar código en etapas posteriores.

Actividades de Seguridad: *Principios de Diseño Seguro, Revisiones de Diseño, Modelado de Amenazas.*

2.4. Etapa de Implementación.

Se escribe código y se integran componentes pre-construidos en función del diseño establecido. Integrar las actividades de seguridad a esta etapa minimiza la cantidad de bugs a repararse en etapas posteriores.

Actividades de Seguridad: *Técnicas de Programación defensiva, Análisis Estático, Revisión de Código entre pares.*

2.5. Etapa de Pruebas.

Comparando el resultado de la implementación con los requerimientos preestablecidos, se detecta fallos, se sugiere correcciones y ajustes. ***Nunca debe relegarse las consideraciones de seguridad a esta etapa***. Las pruebas deben servir como punto de verificación y control adicional.

Actividades de Seguridad: *Pruebas de caja negra y caja blanca, Auditorías de Código por 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.

Actividades de Seguridad: *Hardenizado de Sistemas Operativos, 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 bugs. Se corrigen errores e implementan actualizaciones.

Actividades de Seguridad: *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	Simplificado
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 Requerimient. Seg. Riesgo usuarios y arquitect.	Fuentes de Requerimientos de Seguridad. Casos de abuso. Matriz RTM.	Requerimientos de Privacidad y Seguridad. Calidad y bugs aceptables. Análisis de Riesgos Priv/Seg	Evaluar riesgos al sistema. Elegir y documentar controles de seguridad.	Requerimientos de seguridad. Análisis de riesgos.
Diseño	Revisión Diseño Seguro. Revisión Seg. Externa. Análisis Riesgos Diseño.	Evaluación de Superficie de Ataque. Modelados de Amenaza.	Requerimientos Diseño Reduc. Superficie Ataque Modelado de Amenazas	Diseñar arquitectura seg. Ingeniería Seg. + Controles Documentación de Seg.	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 Inseg. 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 Seg. Gestión Resultado	Análisis Dinámico. Fuzz Testing. Revisión Sup. De Ataque.	Evaluar seg. Del sistema Autorizar sistema de info.	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 Seg Certificación de Release y archivado	Ver Preparación operativa. Gestión de configuración.	Hardenizado de sistema operativo y servicios.
Mantenimiento	Gestión de rta. a incidentes. Parches de seguridad. Revisiones de permisos.	Monitoreo. Rta. a incidentes. Backups y Recup.	Ejecutar plan de respuesta a incidentes.	Monitoreo continuo	Respuesta a incidentes. Monitoreo de Seg. 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 que llega a sistemas productivos.*** *También puede ayudar a acortar los plazos para la puesta en producción y aumenta 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 la criticidad de la aplicación, puede aumentar la intensidad y sofisticación de los ataques.

4.2. Algún ataque va a funcionar

Se debe suponer que **algún ataque va a funcionar**. Ya sea por errores de diseño o bugs en la implementación o por vulnerabilidades en la infraestructura de la que depende la aplicación.

Considerar la posibilidad real de que un atacante evadirá los controles de seguridad establecidos. Se deberá aplicar principios de desarrollo seguro para **minimizar el impacto de los ataques exitosos**. *Cuando un atacante obtenga acceso al sistema, ¿Cómo minimizar el impacto contra la organización?*

4.3. Privacidad de los Usuarios

La información personal de sus usuarios, una vez filtrada no volverá a ser privada. Una filtración de datos puede afectar los procesos y la reputación de la organización. Una vez tomada la decisión de almacenar datos privados, debe asumirse la responsabilidad de proteger su confidencialidad 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 Equipo de Seguridad

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

Los responsables de seguridad deberán 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 comprendiendo 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

Los 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 almacenada en sistemas informáticos o 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 abrir en una vulnerabilidad y ser explotado. 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

Habrá que corregir bugs. 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 para corregir bugs afecta la "*ventana de vulnerabilidad*", el plazo durante el que se conoce una falla de seguridad del sistema sin corregir.

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 reforzarlos ajustando el diseño o implementando controles de seguridad.

6.4. Seguridad por Defecto.

Los usuarios no reconfiguran los parámetros de seguridad. Se debe establecer los controles en su configuración más segura aceptable. Y eventualmente ofrecer la posibilidad de deshabilitar algunos de ellos. Se recomienda desalentar configuraciones inseguras.

6.5. Mantener la Usabilidad.

***El buen software es invisible*.** 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 los recursos de la aplicación**. En caso de que se decidiera publicar ciertos recursos, debe fundamentarse la decisión en base a reglas de la organización. Y debería hacerse un análisis de los riesgos asociados a la decisión.

6.7. Principio de Mínimo Privilegio

Establece que se debe **asignar solo los permisos necesarios y suficientes** a un componente o usuario para realizar una acción específica. Se asigna los permisos **lo más tarde posible y se deben revocar inmediatamente** 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 seguirían 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, 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 obscuridad* 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 clasifica amenazas utilizando criterios STRIDE, DREAD, OCTAVE, Etc. Rankeando 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 corrección de vulnerabilidades. Minimiza significativamente la cantidad de vulnerabilidades que llegan 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 incorporarse 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 Bugs

Se recomienda establecer un criterio de rango de bugs, 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 bugs. Siempre se debe verificar una corrección válida y que no haya producido nuevos bugs.

7.5. Prudencia al Confiar en Terceros.

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

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. Debe ejecutarse en un equipo confiable y asegurado, **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)
```

***Defiende de** ataques de inyección, XSS (Cross Site Scripting).*

8.2. Encodear 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)
```

***Defiende de** 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.
```

***Defiende de** baja Mantenibilidad de 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 framework 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 framework, antes de intentar implementar los propios. Siempre ofrecer una funcionalidad de cierre de sesión y un tiempo de terminación automática.

***Defiende de** bypass de autenticación, robo de sesiones, robo y destrucción de datos personales.*

8.5. Control de Accesos.

Unificar los controles de acceso para todo el sitio. La verificación de controles de acceso debe ejecutarse en el servidor. Limitar la cantidad de acciones permitidas por usuario para cierto tiempo. La aplicación solo debe tener permisos de acceso a archivos estrictamente necesarios.

***Defiende de** 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 durante la ejecución del código. En caso de producirse una condición de error no se debe revelar información del funcionamiento interno del sistema a usuarios.

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

Defiende de *revelar información de 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.

Defiende de *ataques de ejecución de código, carga de malware, defacements.*

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 productivas 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.
```

Defiende de *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, framework, 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.

Continuado: Ataques Comunes: OWASP Top 10

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 bugs, 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 bugs 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 una IDE, compilador, intérprete o herramienta especializada, puede ofrecer una excelente medida para prevenir bugs. 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 bugs 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

Dada una correcta verificación durante la etapa de pruebas, 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

Quitar componentes innecesarios. Configurar correctamente los necesarios. Activar componentes de seguridad. 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. Pero requieren mantenimiento adicional y pueden impactar sobre el rendimiento del sistema.

Se recomienda un **esquema de particiones que separe** el sistema operativo, servicis, 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, bugs 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 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 garantizar la integridad y establecerse mecanismos de custodia para garantizar la confidencialidad.

Si se necesita descartar 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, **debe validarse y sanitizarse todos los datos importados.**

13. Referencias

OWASP: Guía Para Construir Aplicaciones y Servicios Seguros
https://www.scribd.com/document/112624664/OWASP-Development-Guide-2-0-1-Spanish-pdf

OWASP: Lista de Verificación de Prácticas de Codificación Segura
https://www.owasp.org/images/c/c8/OWASP_SCP_Quick_Reference_Guide_SPA.doc

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

SEI CERT: Top 10 Secure Coding Practices
https://www.securecoding.cert.org/confluence/display/seccode/Top+10+Secure+Coding+Practices

OWASP: SAMM 1.0
http://www.opensamm.org/downloads/SAMM-1.0.pdf

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

SEI CMU: Secure SDLC Processes
https://www.sei.cmu.edu/reports/05tn024.pdf

IEEE Center for Secure Design: Top 10 Flaws
http://www.computer.org/cms/CYBSI/docs/Top-10-Flaws.pdf

SAFECODE: Software Assuranse, Overview of Current Industry Best Practices
https://www.safecode.org/publication/SAFECode_BestPractices0208.pdf

SAFECODE: Fundamental Practices for Secure Software Development
https://www.safecode.org/publication/SAFECode_Dev_Practices1108.pdf

SAFECODE: Tactical Threat Modeling
https://www.safecode.org/wp-content/uploads/2017/05/SAFECode_TM_Whitepaper.pdf

O'REILLY: Secure Coding Principles and Practices
http://shop.oreilly.com/product/9780596002428.do

Cristian Borghello: Metodologías de Desarrollo Seguro
https://www.youtube.com/watch?v=eMs2fErek_c

Jim Manico: dotSecurity 2017, Secure SDLC
https://www.youtube.com/watch?v=M7qMP3C5bkU

Maurice Dawson: Secure Software Development Life Cycle
https://es.slideshare.net/drdawson/secure-software-development-life-cycle

Mozilla: Web Security
https://developer.mozilla.org/en-US/docs/Web/Security

NIST: SP800-64 Security Considerations in the SDLC
http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-64r2.pdf

NIST: SP800-27 Engineering Principles for IT Security
http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-27ra.pdf

US-CERT: Build Security In
https://www.us-cert.gov/bsi

Microsoft: Security Development Lifecycle
https://www.microsoft.com/en-us/sdl/

Microsoft: The Security Development Lifecycle
https://download.microsoft.com/download/8/1/6/816C597A-5592-4867-A0A6-A0181703CD59/Microsoft_Press_eBook_TheSecurityDevelopmentLifecycle_PDF.pdf

Apple: Secure Coding Guide
https://developer.apple.com/library/content/documentation/Security/Conceptual/SecureCodingGuide/Introduction.html

CWE/SANS: Top 25 Most Dangerous Software Errors
http://cwe.mitre.org/top25/

David Wheeler: Secure Programming HOWTO
https://www.dwheeler.com/secure-programs/Secure-Programs-HOWTO/

ISC2: Certified Secure Software Lifecycle Professional
https://www.isc2.org/Certifications/CSSLP

CSCIA: Secure Programming
http://csc.uis.edu/center/courses/secure_coding.html

UTN: Programación Segura
http://sceu.frba.utn.edu.ar/wp-content/uploads/2017/04/PROGRAMACI%C3%93N-SEGURA.pdf

Best Práctices for Code Review
https://smartbear.com/learn/code-review/best-practices-for-peer-code-review/

SEU/UTN: OWASP Desarrollo Seguro
http://seu.sanfrancisco.utn.edu.ar/evento/curso-online-owasp-desarrollo-seguro-practica-orientada-a-prevencion-de-owasp-top-10-48

AMU/APU: Application Security
http://www.apus.edu/z/course-syllabus/ISSC411.pdf

SCIPP: Secure Web Application Development Awaraness
http://www.scippinternational.org/wp-content/uploads/2016/03/scipp-swada-gap-new.pdf

Harvard: Secure Software Development
https://canvas.harvard.edu/courses/26673/assignments/syllabus

Berkeley: Secure Coding Practice Guidelines
https://security.berkeley.edu/secure-coding-practice-guidelines

University of Texas: Minimum Security Standards for Application Development and Administration
https://security.utexas.edu/policies/standards_application

IEEE Computer Society: Secure Software Coding
https://www.computer.org/web/education/secure-software-coding

Cybrary: Sunny Wear, Secure Coding
https://www.cybrary.it/course/secure-coding/

SANS: A Security Checklist for Web Application Design
https://www.sans.org/reading-room/whitepapers/securecode/security-checklist-web-application-design-1389

Synopsys: BSIMM 7
https://www.synopsys.com/content/dam/synopsys/bsimm/reports/BSIMM7.pdf

Pluralsight: CSSLP Secure Software Concepts
https://app.pluralsight.com/library/courses/csslp-secure-software-concepts/

ISASecure: Security Development Lifecycle Assessment v3.0
http://www.isasecure.org/en-US/Documents/Authentication-Required-Specifications/SDLA-312-Sec-Dev-Lifecycle-Assess(v3_0)-(1)