JOSÉ MANUEL ORTEGA CANDEL

DESARROLLO SEGURO
EN INGENIERÍA DEL SOFTWARE

APLIACIONES SEGURAS CON ANDROID, NODEJS, PYTHON Y C++

CONTENIDOS WEB

Marcombo Tecnología, Ciencia y Formación
1945-2020 75 años

alphaeditorial

Desarrollo seguro en ingeniería del software

Aplicaciones seguras con Android, NodeJS, Python y C++

José Manuel Ortega Candel

Desarrollo seguro en ingeniería del software

Aplicaciones seguras con Android, NodeJS, Python y C++

José Manuel Ortega Candel

Marcombo 1945-2020 75 años

alphaeditorial

ALPHA EDITORIAL
ALFAOMEGA COLOMBIANA S.A.
Calle 62 No.20-46 esquina, Bogotá
Teléfono (57-1) 746 0102 Fax: (57-1) 210 0122
cliente@alfaomegacolombiana.com
www.alfaomega.com.co
www.alpha-editorial.com

Primera edición: Madrid, 2020
Bogotá, 2020

Correctores: Mónica Muñoz y Beatriz García
Directora de producción: M.a Rosa Castillo
Diseño de la cubierta: ArteMio
Maquetación: xpress.cat

ISBN: 978-958-778-638-5 (Colombia)
ISBN: 978-84-267-2800-5 (España)

Hecho en Colombia
Printed and made in Colombia

Este libro va dedicado a aquellos que me han seguido,
me siguen y me seguirán, no importa cuál sea el camino escogido,
en algún lugar nos encontraremos.

Índice

Capítulo 1. Introducción al desarrollo seguro

La necesidad de desarrollar aplicaciones seguras y, por tanto, tener en cuenta la seguridad en las metodologías de desarrollo es tan importante como tenerla en cuenta para la construcción de edificios o barcos. Desarrollar aplicaciones sin considerar la seguridad es como fabricar un barco sin botes salvavidas o sin los botes necesarios para salvar a todas las personas a bordo en caso de una catástrofe como la del Titanic.

Actualmente, las nuevas tecnologías son un orquestador fundamental en la sociedad de la información, lo cual incluye a las personas y los nuevos tipos de relaciones que se establecen entre ellas. Esta dependencia se halla presente en múltiples escenarios y contextos, desde actividades comerciales y sociales hasta el ámbito militar o la gestión de las infraestructuras críticas nacionales. Ello ha provocado la necesidad creciente de que la tecnología se comporte como se espera, esto es, de acuerdo con sus especificaciones, y de que implemente los mecanismos de seguridad apropiados desde el punto de vista de la confidencialidad, privacidad y seguridad de los datos.

Es cierto que algunos escenarios demandan unos requisitos de seguridad más rigurosos que otros. Sin embargo, la interdependencia entre sistemas de información en un contexto global como el actual nos sugiere la necesidad de una aproximación global, lo que hace de la confiabilidad en la tecnología un concepto de aplicación universal y no particular a un campo específico.

A pesar de ello, se ha demostrado ampliamente que la tecnología actual no ha alcanzado el grado de madurez y seguridad requerido, pues es vulnerable a una gran cantidad de amenazas. Según estudios realizados, cerca del 90 % de los incidentes de seguridad del software están causados por atacantes que han explotado fallos conocidos en él. Además, en un análisis de 45 aplicaciones de negocio electrónico, se mostró que el 70 % de los fallos del software estaban relacionados con el diseño. De media, un ingeniero de software experimentado y capacitado introduce un fallo por cada nueve líneas de código. Aproximadamente, un millón de líneas de código podrían tener de mil a cinco mil fallos, los cuales podrían ser el origen de alrededor de cien vulnerabilidades en producción. Además, se ha demostrado que el software se degrada con el tiempo, y lo que vale en un momento en concreto puede no valer al cabo de seis meses.

En este sentido, la ingeniería de software de desarrollo seguro es una disciplina de la ingeniería del software en la que se busca proporcionar garantías objetivas respecto a la seguridad del software desarrollado. Para ello, se aplican

mecanismos y se producen evidencias durante el proceso que garantizan que el software contiene las propiedades de seguridad que se le requieran.

En particular, y partiendo del axioma "la seguridad absoluta no es alcanzable", el software seguro es capaz de minimizar el impacto de la mayoría de los ataques, tolerar aquellos que no pueda resistir, y recuperarse rápidamente y con el menor impacto de aquellos otros que no pueda tolerar.

El campo de la investigación en ingeniería de software seguro se centra en desarrollar métodos eficaces y herramientas que permitan al desarrollador, durante el ciclo de vida del desarrollo de software, la implementación segura desde las primeras fases de análisis hasta las etapas de más largo plazo de desarrollo.

Uno de los grandes retos actuales para la aplicación de cualquier técnica orientada a mejorar la calidad y la seguridad del software es el cambio de mentalidad en el desarrollador, así como, en muchos casos, la compleja integración de estas técnicas en los procesos existentes, con los costes que ello conlleva. Los hechos nos demuestran que resulta urgente mejorar los procesos de ingeniería del software en pro de unos sistemas de información más robustos y seguros que aporten la confianza necesaria a la sociedad.

El desarrollo seguro es una parte importante de la seguridad informática, englobado dentro del ámbito de la prevención. Podríamos definir que un programa seguro es aquel capaz de seguir realizando las funciones para las que ha sido creado en todo momento, y capaz de evitar que la existencia de errores en él pueda ser utilizada como puente para la realización de acciones que pongan en peligro la integridad, confidencialidad o disponibilidad del resto de elementos del sistema en el que se está ejecutando.

Por tanto, la programación segura engloba el conjunto de técnicas, normas y conocimientos que permiten crear programas que no puedan ser modificados de forma ilegítima con fines maliciosos y que estén carentes de fallos que puedan comprometer la seguridad del resto de elementos del sistema con el que interactúan.

Los objetivos de la utilización de prácticas de software seguras son las siguientes:

- Los fallos explotables y otros puntos débiles se eliminan en la mayor medida posible.
- La probabilidad de que los desarrolladores puedan producir errores y vulnerabilidades explotables o puertas traseras en el software se reduce o elimina en gran medida.

- El software es resistente y tolerante a posibles ataques para apoyar el cumplimiento de la misión de la organización.

Uno de los principales objetivos radica en ayudar a desarrolladores de software a implementar medidas preventivas que ayuden a mitigar o reducir lo máximo posible errores comunes de programación y que pueden ser aprovechados por un posible atacante para comprometer una aplicación.

El libro presenta gran interés para cualquier desarrollador responsable de una página web o un administrador de grandes portales corporativos, así como para analistas y programadores de una empresa de las denominadas *software factories*. El lector estará en disposición de definir procedimientos y políticas de desarrollo de código dentro de una misma organización, y también de establecer los controles necesarios para el desarrollo de software seguro.

1.1 Propiedades del software seguro

Si bien con las metodologías y los estándares relacionados con el desarrollo y la construcción de software se busca mantener altos niveles de confiabilidad y control de la solución informática, la seguridad informática y sus principios de diseño seguro no suelen ser parte formal u obligatoria de dichos estándares.

Cuando una aplicación transmite o almacena información confidencial, la aplicación es responsable de garantizar que los datos almacenados y transferidos estén cifrados y no puedan obtenerse, alterarse o divulgarse fácilmente de forma ilícita. En general, se suele decir que el objetivo fundamental de la seguridad informática se basa en preservar los siguientes puntos:

- **Confidencialidad**. El software debe asegurar que cualquiera de sus características, los activos que administra y su contenido son accesibles solo para las entidades autorizadas e inaccesibles para el resto. El acceso a la información está limitado a usuarios autorizados. Los datos deben protegerse de la observación o divulgación no autorizadas tanto en tránsito como almacenadas.
- **Integridad**. El software y los activos del sistema solo pueden ser modificados por usuarios autorizados. Esta propiedad se debe preservar durante el desarrollo del software y su ejecución. Los datos han de protegerse al ser creados, alterados o borrados maliciosamente por atacantes no autorizados.

- **Disponibilidad**. El software debe estar operativo y ser accesible a sus usuarios autorizados siempre que se requiera, así como ha de desempeñarse con una *performance* adecuada para que los usuarios puedan realizar sus tareas de forma correcta y dar cumplimiento a los objetivos de la organización que lo utiliza. El acceso a los activos en un tiempo razonable está garantizado para usuarios autorizados. Los datos deben encontrarse disponibles para los usuarios autorizados, según sea necesario.

Para las entidades que actúan como usuarios, se requieren dos propiedades adicionales:

- **Trazabilidad**. Todas las acciones relevantes relacionadas con la seguridad de una entidad que actúa como usuario se deben registrar y trazar con el objetivo de disponer de datos para la realización de auditorías; de esta forma, la trazabilidad debe ser posible tanto durante la ocurrencia de las acciones registradas como *a posteriori*.
- **No repudio**. Constituye la habilidad de prevenir que una entidad que actúa como usuario desmienta o niegue la responsabilidad de acciones que han sido ejecutadas.

Estas propiedades básicas son las más utilizadas normalmente para describir la seguridad de los sistemas y aplicaciones. Un ataque con éxito de **inyección de SQL** en una aplicación para extraer información de identificación personal de su base de datos sería una violación de la propiedad de **confidencialidad**. El éxito de un ataque **cross-site scripting** (XSS) en contra de una aplicación web podría dar lugar a una violación tanto en su **integridad** como en su **disponibilidad**. Un ataque con éxito de desbordamiento de búfer que inyecta código con el objetivo de obtener y modificar la información de usuarios sería una violación de las cinco propiedades básicas de seguridad.

1.2 Principios de diseño seguro de aplicaciones

En consecuencia, los efectos de vulnerar la seguridad del software se pueden describir en términos de los efectos sobre estas propiedades fundamentales. A continuación se sugieren una serie de principios orientados al diseño seguro de aplicaciones:

1.2.1 Minimizar el área de la superficie de ataque

Cada característica que se añade a una aplicación incrementa su complejidad y aumenta también el riesgo de aplicación en conjunto. Una nueva característica implica un nuevo punto de ataque. Uno de los factores clave para reducir el riesgo de una aplicación recae en la reducción de la superficie de ataque. Pueden eliminarse posibles puntos de ataque si se deshabilitan módulos o componentes innecesarios para la aplicación; por ejemplo, si la aplicación no utiliza el almacenamiento en caché de resultados, sería recomendable deshabilitar dicho módulo. De esta manera, si se detecta una vulnerabilidad de seguridad en ese módulo, la aplicación no se verá amenazada.

1.2.2 Seguridad por defecto

Existen muchas maneras de entregar una experiencia *out of the box* ("lista para usar") a los usuarios. Sin embargo, por defecto, la experiencia debe ser segura, mas facilitar, no obstante, la capacidad de reducción del nivel de seguridad si el usuario lo cree necesario.

Del lado de los desarrolladores, constituye una práctica habitual utilizar opciones de configuración de seguridad reducidas para evitar que dichas configuraciones compliquen el desarrollo. Si, para su implementación, la aplicación requiere de características que obligan a reducir o cambiar la configuración de seguridad predeterminada, se recomienda estudiar sus efectos y consecuencias, para lo que hay que realizar pruebas específicas de auditoría de seguridad.

1.2.3 Privilegios mínimos

Según el principio del mínimo privilegio, se recomienda que las cuentas de usuario tengan la mínima cantidad de privilegios necesarios. Asimismo, se aconseja que los procesos se ejecuten únicamente con los privilegios necesarios para completar sus tareas. De esta manera, se limitan los posibles daños que podrían producirse si se ve comprometido el proceso.

Si un atacante llegase a tomar el control de un proceso en un servidor o comprometiese una cuenta de usuario, los privilegios concedidos determinarán, en gran medida, los tipos de operaciones que podrá llegar a realizar dicho atacante; por ejemplo, si cierto servidor solo requiere acceso de lectura a la tabla de una base de datos, bajo ninguna condición deberían darse privilegios administrativos a los usuarios si no resultan necesarios.

1.2.4 Validación de datos de entrada

Se ha de garantizar que la aplicación sea robusta ante todas las posibles formas de entrada de datos, ya sean proporcionados por el usuario, por la infraestructura, por entidades externas o por bases de datos.

Una premisa fundamental reside en no confiar en los datos que el usuario pueda introducir, ya que este tiene todas las posibilidades de manipularlos. La debilidad de seguridad más común en aplicaciones es la falta de validación apropiada de las entradas del usuario o del entorno. Esta debilidad origina casi todas las principales vulnerabilidades en las aplicaciones, tales como inyecciones de código, inserción de secuencias de comandos, ataques al sistema de archivos o desbordamientos de memoria.

Las aplicaciones deben validar todos los datos introducidos por el usuario antes de realizar cualquier operación con ellos. La validación podría incluir el filtrado de caracteres especiales o el control de la longitud de los datos introducidos. Esta medida preventiva protege a la aplicación de usos incorrectos accidentales o de ataques deliberados por parte de usuarios.

1.2.5 Defensa en profundidad

Con "defensa en profundidad", nos referimos a definir una estrategia de seguridad estándar en la que se establezcan varios controles de defensa en cada una de las capas y los ssubsistemas de la aplicación. Estos puntos de control ayudan a garantizar que solo los usuarios autenticados y autorizados puedan obtener el acceso a la siguiente capa y a sus datos.

1.2.6 Control seguro de errores

Es necesario controlar las respuestas cuando se produce algún error y no mostrar en ellas información que pudiera ayudar al atacante a descubrir datos acerca de cómo ha sido desarrollada la aplicación o cómo funcionan sus procedimientos. La información detallada de los errores producidos no debería ser mostrada al usuario, sino que tendría que ser enviada al fichero de log correspondiente. Una simple página de error 403 indicando un acceso denegado puede decir a un escáner de vulnerabilidades que un directorio existe y puede proporcionar a un atacante información que le permita realizar un mapa de la estructura de directorios de la aplicación.

1.2.7 Separación de funciones

Un punto importante consiste en la separación de funciones entre los distintos perfiles de la aplicación; por ejemplo, en una tienda de subastas online, un usuario vendedor pone en subasta un producto. Si la separación de funciones se encuentra correctamente implementada, este mismo usuario no debe poder participar en la puja por el artículo. Por otra parte, determinados roles deben contar con un nivel de confianza más elevado que los usuarios normales, como en el caso del administrador, que posee privilegios avanzados, como administrar al resto de usuarios del sistema o configurar políticas de contraseñas, así como definir los diferentes parámetros de la aplicación.

1.2.8 Evitar la seguridad por oscuridad

La seguridad de una aplicación no debería depender del secreto o confidencialidad de su diseño o implementación. Si se intentan ocultar secretos mediante el uso de nombres de variables engañosos o de ubicaciones de archivos no habituales, no se estará mejorando la seguridad. La seguridad basada en la oscuridad es un control de seguridad débil, especialmente si se trata del único control. Esto no significa que mantener secretos constituya una mala idea; significa que la seguridad de los sistemas clave no debería basarse, exclusivamente, en mantener detalles ocultos.

Por ejemplo, la seguridad de una aplicación no debería basarse en mantener en secreto el conocimiento del código fuente. La seguridad ha de fundamentarse en muchos otros factores, incluyendo políticas de contraseñas, diseño de una arquitectura sólida tanto a nivel de aplicación como a nivel de comunicaciones de red y realización periódica de controles de auditoría. Un ejemplo práctico es Linux, cuyo código fuente es open source y está disponible para todo el mundo y, aun así, se trata de un sistema operativo seguro y robusto.

1.3 Análisis de requisitos de seguridad

La seguridad debería ser un aspecto de alta prioridad cuando estamos desarrollando y probando aplicaciones que manejan datos confidenciales a nivel de autenticación y autorización. El análisis de los requisitos de seguridad debe ser parte de la primera fase de análisis de los requisitos de cada proyecto de aplicaciones móviles. La siguiente lista lo puede ayudar con el análisis de los requisitos de seguridad:

- Identificar los posibles roles de usuario, así como sus limitaciones y permisos dentro de la arquitectura (app y backend).
- Identificar qué tipo de enfoques y herramientas de pruebas de seguridad se requieren para lograr un buen nivel de seguridad.
- Identificar el impacto que tienen las funcionalidades a nivel de usuario en la seguridad a nivel frontend y backend.

La mayoría de los fallos de seguridad de aplicaciones se pueden prevenir mediante la integración de los procesos y buenas prácticas de seguridad desde las primeras etapas de desarrollo. La planificación de una estrategia inicial de diseño de aplicaciones y mantenimiento de la seguridad en mente todo el tiempo permite reducir las posibilidades de los riesgos de seguridad que surgen durante las últimas etapas de desarrollo de aplicaciones.

Capítulo 2. Aspectos fundamentales de desarrollo seguro

Resulta importante hacer entender que la seguridad debe formar parte del ciclo de desarrollo como algo integrado dentro de las herramientas de integración continua, no como una fase separada, sino como parte integral del desarrollo. El objetivo, al final, es tratar de minimizar los riesgos que pueda suponer la seguridad de las aplicaciones.

Una buena práctica sería poner toda la lógica de negocio posible en la parte de servidor/backend, para evitar que alguien pueda hacer ingeniería inversa en la aplicación y averiguar determinados datos. En el caso de que haya que introducir datos sensibles, se deberían encriptar para evitar ataques.

Cuando se habla de seguridad, los atacantes tienen una ventaja sobre los desarrolladores. Esta se expone en cuatro principios que la ilustran muy bien:

- Quien defiende debe preocuparse por proteger todos los puntos; el atacante seleccionará solo uno: el más débil.
- Los encargados de la defensa van a procurar defenderse de ataques conocidos; el atacante probará nuevas formas de llevar adelante sus ataques.
- Quien se encarga de defender debe estar en constante estado de vigilancia.
- El defensor ha de respetar reglas; el atacante, normalmente, juega con sus propias reglas.

Cabe resaltar que un atacante necesita solamente un pequeño error, una vulnerabilidad, para lograr su cometido. Si dentro de la empresa se descuidan los temas de seguridad por acelerar la operatividad del negocio, podemos estar dejando la puerta abierta a que se comprometa la seguridad de la información. Ser responsables con los procesos constituye la mejor defensa, y no está de más preguntarse si es mejor invertir unas semanas más en desarrollo que perder reputación y dinero en un instante por un incidente de seguridad.

Controlar un sistema resulta parte fundamental para garantizar un mejoramiento continuo. Por esta razón, vamos a ver algunos controles que deberían tenerse en cuenta para que una aplicación garantice la integridad y disponibilidad de la información y, además, se pueda mejorar progresivamente.

En particular, para aquellos que están directamente relacionados con el desarrollo, la responsabilidad es mayor, pues, finalmente, son quienes atienden los requerimientos e implementan las soluciones.

Por esta razón, se recomienda tener en cuenta algunos controles en sus desarrollos, para prevenir escenarios como el **cross-site scripting (XSS)**, las inyecciones de código, la ejecución de códigos maliciosos y el **cross-site request forgery**. Posteriormente, revisamos el proyecto de OWASP, que publica un top 10 de vulnerabilidades, y donde se presentan los riesgos más críticos de aplicaciones web, junto con la forma de prevenirlos.

Los controles implementados dentro de los desarrollos deben validar la entrada, el procesamiento y la salida de los datos, de forma que se cumpla con los niveles de confidencialidad, integridad y disponibilidad.

De esta forma, deben implementarse **controles de detección**, que ayudan a identificar los accesos no autorizados o entradas no válidas; **controles preventivos**, diseñados para evitar la entrada de datos no autorizados o no válidos, y **controles proactivos**, que permiten probar los componentes de forma continua, a medida que se desarrollan, mediante pruebas unitarias y de integración.

Los **controles de detección** aseguran que las operaciones se hagan con la exactitud requerida y, además, que las consultas que se realicen sobre bases de datos u otros sistemas sean consistentes. Operaciones como la combinación de archivos, la modificación de datos, la actualización de bases de datos o los mantenimientos de sistemas son operaciones sensibles que deberían tener este tipo de controles.

Los **controles preventivos** deben enfocarse en que la información ingresada sea precisa y, por lo tanto, en que aquello que se registre en el sistema sea adecuado. Dentro de estos controles, se pueden tener validaciones mediante el uso de expresiones regulares, comprobar la integridad de los campos, verificar que no exista duplicidad en los datos, establecer rangos de validación e incluso comprobar que los valores de los campos sean razonables de acuerdo con unos criterios predefinidos.

2.1 Controles proactivos

Independientemente de los controles implementados, resulta muy importante, además, definir un adecuado plan de pruebas que incluya, asimismo, las pruebas de aceptación por parte de los usuarios finales, donde se evalúa la funcionalidad de la aplicación.

Aparte de los usuarios finales, deberían considerarse recursos del equipo de desarrollo para realizar pruebas de funcionalidad del código y de integración.

Incluso si la aplicación es muy crítica para el negocio, se podría considerar la alternativa de que un equipo externo al equipo de desarrollo también complemente con otras pruebas a las ya realizadas por el equipo de desarrollo.

Para lograr un software seguro, los desarrolladores deben contar con el respaldo y la ayuda de la organización para la que escriben el código, cosa que, a veces, no sucede. A medida que los desarrolladores de software crean el código que compone una aplicación, se necesita adoptar y practicar una amplia variedad de técnicas de codificación segura. Todas las capas que componen una aplicación, tales como la interfaz de usuario, la lógica de negocio, el controlador o la base de datos, deben desarrollarse teniendo la seguridad en mente. La mayoría de los desarrolladores no han aprendido sobre la codificación segura o la criptografía en su etapa de formación y, en ocasiones, tampoco se lo han exigido.

Los lenguajes y frameworks que los desarrolladores usan para construir aplicaciones a menudo carecen de controles centrales críticos o son inseguros por defecto de alguna manera. También, en honor a la verdad, es muy raro que las organizaciones brinden a los desarrolladores requisitos prescriptivos que los guíen por el camino del software seguro e, incluso, cuando lo hacen, puede haber errores de seguridad inherentes a los requisitos y diseños. Por ello, se ha de intentar ser proactivo desde los inicios del diseño del proyecto que se realice, teniendo en cuenta, en principio, la seguridad.

2.2 OWASP (Open Web Application Security Project)

OWASP (Open Web Application Security Project) nos provee de una serie de recursos basados, sobre todo, en guías y herramientas, para que nuestros proyectos web sean lo más seguros posible, tanto desde el punto de vista de desarrollo seguro como de la evaluación de la seguridad de estos.

Para los desarrolladores y auditores de seguridad, ofrece una gran cantidad de recursos para ayudar a llevar a cabo un ciclo de vida de desarrollo de software seguro (S-SDLC, Secure Software Development Life Cycle), así como una variedad de recursos, como guías y herramientas, para evaluar la seguridad de las aplicaciones web.

Entre los principales objetivos de la OWASP, podemos destacar:

- Proporcionar a los desarrolladores un conjunto de buenas prácticas, para que las aplicaciones sean lo más seguras posible.

- Proporcionar a desarrolladores y profesionales de seguridad recursos para asegurar aplicaciones; en concreto, para aplicaciones móviles, surgió el proyecto OWASP Mobile Security Project.

El objetivo del proyecto es intentar ayudar a los desarrolladores a comprender qué, por qué, cuándo, dónde y cómo probar aplicaciones web. Los desarrolladores pueden usar este framework como plantilla para construir sus propios programas de prueba o para evaluar los procesos de otros desarrolladores. En la guía se describen en detalle tanto el framework de pruebas en general como las técnicas requeridas para llevarlo a la práctica.

El proyecto de seguridad en aplicaciones web (OWASP) es una comunidad abierta dedicada a facultar a las organizaciones a desarrollar, adquirir y mantener aplicaciones que pueden resultar confiables. Esta organización nos presenta los 10 riesgos más comunes, entre los que podemos destacar:

- **Inyección**. Los errores de inyección, tales como SQL y LDAP, ocurren cuando datos no confiables son enviados a un intérprete como parte de un comando o consulta. Los datos hostiles del atacante pueden engañar al intérprete a la hora de ejecutar comandos no intencionados o acceder a datos no autorizados.
- **Pérdida de autenticación y gestión de sesiones**. Las funciones de la aplicación relacionadas con la autenticación y gestión de sesiones son, frecuentemente, implementadas de manera incorrecta, lo que permite a los atacantes comprometer contraseñas, claves, *token* de sesiones o explotar otros errores de implementación para asumir la identidad de otros usuarios.
- Secuencia de comandos en sitios cruzados (XSS). **Los errores XSS ocurren** cada vez que una aplicación toma datos no confiables y los envía al navegador web sin una validación y codificación apropiada. XSS permite a los atacantes ejecutar una secuencia de comandos en el navegador de la víctima, los cuales pueden secuestrar las sesiones de usuario, destruir sitios web o dirigir al usuario hacia un sitio malicioso.
- **Referencia directa insegura a objetos**. Una referencia directa a objetos ocurre cuando un desarrollador expone una referencia a un objeto de implementación interno, tal como un fichero, directorio o base de datos. Sin un chequeo de control de acceso u otra protección, los atacantes pueden manipular estas referencias para acceder a datos no autorizados.

- **Configuración de seguridad incorrecta**. Una buena seguridad requiere tener definida e implementada una configuración segura para la aplicación, marcos de trabajo, servidor de aplicación, servidor web, base de datos y plataforma. Todas estas configuraciones deben ser definidas, implementadas y mantenidas ya que, por lo general, no son seguras por defecto. Esto incluye mantener todo el software actualizado, incluidas las librerías de código utilizadas por la aplicación.
- **Exposición de datos sensibles**. Muchas aplicaciones web no protegen adecuadamente datos sensibles, tales como números de tarjetas de crédito o credenciales de autenticación. Los atacantes pueden robar o modificar tales datos para llevar a cabo fraudes, robos de identidad u otros delitos. Los datos sensibles requieren de métodos de protección adicionales, tales como el cifrado de datos, así como también de precauciones especiales en un intercambio de datos con el navegador.
- **Ausencia de control de acceso a funciones**. La mayoría de las aplicaciones web verifican los derechos de acceso a nivel de función antes de hacer visible una funcionalidad en la interfaz de usuario. A pesar de esto, las aplicaciones necesitan verificar el control de acceso en el servidor cuando se accede a cada función. Si las solicitudes de acceso no se verifican, los atacantes podrán realizar peticiones sin la autorización apropiada.
- **Falsificación de peticiones en sitios cruzados (CSRF)**. Un ataque CSRF obliga al navegador de una víctima autenticada a enviar una petición HTTP falsificada, incluyendo la sesión del usuario y cualquier otra información de autenticación, incluida automáticamente, a una aplicación web vulnerable. Esto permite al atacante forzar al navegador de la víctima para generar peticiones que la aplicación vulnerable piensa que son peticiones legítimas.
- **Utilización de componentes con vulnerabilidades conocidas**. Algunos componentes tales como las librerías, los frameworks y otros módulos de software casi siempre funcionan con todos los permisos y privilegios. Si se ataca un componente vulnerable, esto podría facilitar la intrusión en el servidor o una pérdida de datos. Las aplicaciones en las que se utilicen componentes con vulnerabilidades conocidas permiten ampliar el rango de posibles ataques.
- **Redirecciones y reenvíos no validados**. Las aplicaciones web, frecuentemente, redirigen y reenvían a los usuarios hacia otras páginas o

sitios web y utilizan datos no confiables para determinar la página de destino. Sin una validación apropiada, los atacantes pueden redirigir a las víctimas hacia sitios de phishing o malware, o utilizar reenvíos para acceder a páginas no autorizadas.

Un proyecto interesante que ofrece OWASP es la aplicación open source **Zed Attack Proxy Project (ZAP)** https://www.zaproxy.org, que nos permite realizar un análisis de todos los datos que se envían y que recibimos a la hora de realizar una navegación de un sitio web. La herramienta se puede descargar desde el repositorio de github: https://github.com/zaproxy/zaproxy/wiki/Downloads

La herramienta soporta análisis automático y manual. El primero es más sencillo y menos eficiente, ya que, si se requiere introducir login en la web, solo va a escanear hasta ese punto. Para utilizar el modo automático, bastaría con introducir la url que deseamos analizar en el apartado "url a atacar" y presionar el botón "Atacar".

Figura 2.1 Modos de funcionamiento de OWASP ZAP.

En el modo manual, ZAP actúa como un proxy de intercepción de las peticiones que se realizan en el navegador, de forma que es capaz de analizar cada una de las url para extraer valores de formularios y parámetros de navegación del usuario. Utilizando esta opción, OWASP ZAP utiliza un proxy propio para capturar las peticiones que realizamos durante una sesión de navegación. Toda la navegación efectuada será monitorizada en la aplicación ZAP y se irá mostrando en el apartado "Sitios". En el panel inferior se ven los resultados obtenidos en forma de alertas.

Figura 2.2 Alertas obtenidas al realizar un análisis sobre un sitio web.

2.3. OWASP Mobile Security Project

El objetivo del proyecto reside en crear conciencia acerca de la seguridad en aplicaciones, identificando algunos de los riesgos más críticos a los que se enfrentan las organizaciones. Pues bien, tanto los desarrolladores de aplicaciones móviles como los auditores pueden aprovecharse también de recursos de este tipo para alcanzar el objetivo de la máxima seguridad posible. Para ello, OWASP nos propone el proyecto Mobile Security Project.

La checklist de OWASP para dispositivos móviles está centrada en la seguridad referente a aplicaciones móviles. Trata de dar a los desarrolladores de aplicaciones un recurso centralizado para mantener las aplicaciones seguras. El objetivo es clasificar los riesgos de seguridad y proveer de controles para reducir el impacto o el riesgo de explotación. OWASP se focaliza en la capa de aplicación, en la aplicación entre la aplicación y los recursos remotos de servicios de autenticación, y en las características específicas de plataformas *cloud*.

El proyecto de seguridad móvil OWASP ofrece a los desarrolladores y equipos de seguridad los recursos para construir y mantener aplicaciones móviles seguras. También permite clasificar los riesgos de seguridad móvil y proporciona controles de desarrollo para reducir su impacto o probabilidad de explotación.

Tal y como podemos encontrar en el top 10 de riesgos en aplicaciones móviles https://www.owasp.org/index.php/Mobile_Top_10_2016-Top_10, la siguiente lista nos indica el top 10 de los controles de seguridad para evaluar en aplicaciones móviles:

- **M1-Improper Platform Usage (uso indebido de plataformas)**. Esta categoría cubre el mal uso de las características de las plataformas móviles o de los fallos de uso en los controles de seguridad. Puede incluir permisos en las plataformas, así como la mala utilización de TouchID,

del llavero u otros controles de seguridad que forman parte del sistema operativo.

- **M2-Insecure Data Storage (almacenamiento de datos inseguro)**. Esta categoría cubre las antiguas categorías M2 y M4 del año 2014 (M2: Weak Server Side Controls; M4: Client Side Injection). Cubre el almacenamiento de datos inseguro y la fuga de datos no deseados.
- **M3-Insecure Communication (comunicaciones inseguras).** Esta categoría cubre un mal establecimiento de conexión o handshake, versiones SSL inseguras o comunicaciones en texto en claro sensibles. Toda comunicación donde se transmitan datos de usuarios debe realizarse mediante un canal seguro. Si la aplicación implementa sesiones de usuario mediante el uso de cookies, se deben proteger todas las conexiones contra el servidor.
- **M4-Insecure Authentication (autenticación insegura)**. Esta categoría incluye nociones de autenticación del usuario final o una mala gestión de la sesión del usuario. Algunos ejemplos son no poder identificar al usuario, con lo que se pierde trazabilidad de las acciones, o no mantener la identidad del usuario cuando se requiere.
- **M5-Insufficient Cryptography (criptografía insuficiente)**. Se hace uso de la criptografía en el código para proteger activos sensibles. Sin embargo, a veces, la criptografía no es suficiente en las aplicaciones, ya que se emplean algoritmos criptográficamente rotos, como MD5 o SHA1. En este punto resulta recomendable usar, al menos, un hash del tipo SHA256, que devuelve un bloque de 256 ante cualquier entrada. Al aumentar los bits de salida del algoritmo, en comparación con el algoritmo MD5, disminuye la posibilidad de encontrar colisiones; por lo tanto, es más seguro.
- **M6-Insecure Authorization (autorización insegura)**. Se refiere a fallos en las autorizaciones en las aplicaciones. Es diferente a problemas de autenticación e identificación de usuarios. Cuando las aplicaciones autorizan, dan acceso a los usuarios a visualizar en la aplicación solo aquello que debe visualizar. Se ha de dotar de permisos al usuario, ya sean de escritura, lectura, ejecución u otros permisos más afines al negocio.
- **M7-Client Code Quality (calidad del código de cliente)**. Aquí podríamos incluir problemas relacionados, como desbordamiento de búfer, vulnerabilidades de cadena de formato y otros varios errores a nivel de código, donde la solución estriba en volver a escribir algo de código, que se ejecuta en el dispositivo móvil.

- **M8-Code Tampering (manipulación de código)**. Esta categoría incluye parchear binarios o modificar recursos locales y memoria dinámica. Un atacante puede directamente modificar el código, cambiar contenidos en la memoria dinámica o incluso reemplazar el sistema de API, que usa la aplicación, o bien cambiar los recursos y datos de la aplicación.
- **M9-Reverse Engineering (ingeniería inversa)**. Esta categoría incluye el análisis de binarios para obtener el código fuente y las librerías. Herramientas como IDA Pro, Hooper y otras de inspección de binarios permitirán al atacante visionar acerca del funcionamiento interno de la aplicación.
- **M10-Extraneous Functionality (funcionalidad extraña)**. En ocasiones, los desarrolladores incluyen funcionalidades ocultas de acceso mediante backdoors u otros controles de seguridad internos, no destinados al entorno de producción; por ejemplo, un desarrollador podría incluir, accidentalmente, una contraseña en un comentario de la aplicación o la posibilidad de desactivar el doble factor de autenticación durante la etapa de *testing*.

2.4 Controles proactivos OWASP

El proyecto OWASP Application Security Verification Standard (ASVS) proporciona una base para probar los controles técnicos de seguridad de aplicaciones web y también proporciona a los desarrolladores una lista de requisitos para un desarrollo seguro.

El estándar proporciona una base para probar los controles técnicos de seguridad de las aplicaciones, así como los controles técnicos de seguridad en el entorno, utilizados para proteger contra vulnerabilidades como **Cross-site scripting (XSS) e inyección de SQL**. Este estándar se puede utilizar para establecer un nivel de confianza en la seguridad de las aplicaciones web.

A continuación comentaremos los principales controles proactivos que OWASP recomienda y que deberían incluirse en cada proyecto de desarrollo de software. Entre la lista de los controles que se deben realizar, por orden de importancia, podemos destacar:

- **Verificación de la seguridad desde las primeras etapas de desarrollo**
- **Validación de las entradas del cliente**
- **Desbordamientos del búfer**
- **Gestión de sesiones**

- **Implementación de controles de acceso**
- **Implementación de controles de identidad y autenticación**
- **Autenticación por múltiples factores**
- **Manejo de errores y excepciones**

2.4.1 Verificación de la seguridad desde las primeras etapas de desarrollo

Las pruebas de seguridad deberían ser parte integral de la práctica de ingeniería de software de un desarrollador. En muchas organizaciones, esto no suele hacerse, ya que las pruebas de seguridad se realizan fuera de los ciclos de pruebas de desarrollo.

Por ello, es importante realizar comprobaciones de seguridad de manera temprana y lo más frecuentemente posible, ya sea mediante pruebas manuales o pruebas automatizadas. Obviamente, esto aumenta los plazos de entrega y lógicamente, por ello, ha de ser tenido en cuenta desde el inicio del diseño de la aplicación.

Como parte del desarrollo, debe plantearse la necesidad de implementar medidas de testing durante la fase de desarrollo. Para ello, OWASP tiene un proyecto de testing del código. Puede considerarse **OWASP ASVS (Application Security Verification Standard Project)** https://github.com/OWASP/ASVS, como una guía para definir los requisitos de seguridad y las pruebas.

2.4.2 Validación de las entradas del cliente

La debilidad más común en la seguridad de las aplicaciones web es la falta de validación adecuada de los datos que provienen del cliente o del entorno antes de utilizarlo. Se debe validar y filtrar cada input de la aplicación, tanto a nivel frontend como backend. Incluso si los datos provienen de su aplicación, son posibles su intercepción y manipulación. Sin la validación correcta de todas las entradas, podríamos exponer nuestra aplicación a ataques, como desbordamientos de búfer e inyección de SQL.

Esta debilidad conduce a casi todas las principales vulnerabilidades en las aplicaciones web, tales como scripting cross site, inyección de SQL, inyección de intérprete, ataques de sistemas de archivos y desbordamientos de búfer. En este punto cabe asegurar que la aplicación satisface los siguientes requisitos de gestión de sesiones de alto nivel:

- Todas las entradas están validadas para ser correctas y utilizables para el propósito previsto.
- Los datos de una entidad externa o cliente nunca deben ser confiables y han de tratarse en todos los casos.

La validación de las entradas se tiene que realizar tanto en el lado del cliente como en el lado del servidor; por ejemplo, la validación JavaScript puede alertar al usuario de que un campo, en particular, debe consistir en números, pero el servidor ha de validar que el campo realmente consiste en números. Por otro lado, no podemos olvidar que la validación de entrada no debe utilizarse como el método principal para prevenir vulnerabilidades del tipo XSS e inyección de SQL, pero contribuye significativamente a reducir su impacto, si se implementa correctamente.

Todos los parámetros que gestiona la aplicación han de ser validados, especialmente cuando son recibidos como datos de entrada por el usuario. De esta forma, **dicha validación se realizará tanto en la parte del cliente como a nivel de servidor**. Dentro de esta validación, se comprobará:

- Si los caracteres incluidos son correctos, no incluyen caracteres HTML o códigos SQL o JavaScript.
- Si el tipo del valor introducido es válido.
- Si su longitud se encuentra entre el valor mínimo y el máximo permitido.
- Si el rango está entre los límites permitidos.

Una gran mayoría de las vulnerabilidades de la aplicación web surgen por no validar correctamente la entrada del usuario. Los usuarios que utilizan una interfaz de usuario no necesariamente introducen esta "entrada" directamente. En el contexto de aplicaciones y servicios web, esto podría incluir los siguientes casos:

- Cabeceras HTTP
- Cookies
- Parámetros GET y POST (incluyendo campos ocultos)
- Subidas de archivos (incluyendo información, como el nombre del archivo)

Existen dos enfoques generales para realizar la validación de la sintaxis de entrada, comúnmente conocida como "lista negra" y "lista blanca":