ANÁLISIS DE METODOLOGÍAS PARA LA IMPLEMENTACIÓN DE UN ESQUEMA DE SEGURIDAD EN EL DESARROLLO DE APLICACIONES ON LINE

JUAN CARLOS AHUMADA MUNAR

UNIVERSIDAD NACIONAL ABIERTA Y A DISTANCIA UNAD

ESCUELA DE CIENCIAS BÁSICAS, TECNOLOGÍA E INGENIERÍA

BOGOTÁ D.C.

2019

ANÁLISIS DE METODOLOGÍAS PARA LA IMPLEMENTACIÓN DE UN ESQUEMA DE SEGURIDAD EN EL DESARROLLO DE APLICACIONES ON LINE

JUAN CARLOS AHUMADA MUNAR

Monografía

Ing. Alexander Larrahondo Núñez

Director Proyecto Trabajo de Grado

UNIVERSIDAD NACIONAL ABIERTA Y A DISTANCIA UNAD

ESCUELA DE CIENCIAS BÁSICAS, TECNOLOGÍA E INGENIERÍA

BOGOTÁ D.C.

2019

CONTENIDO

pág.

LISTA DE TABLAS

pág.

LISTA DE FIGURAS

pág.

GLOSARIO

AMENAZA: causa potencial de un incidente que puede causar daños a un sistema de información o a una organización.

API (APPLICATION PROGRAMMING INTERFACE): es un conjunto de especificaciones (funciones, métodos, procedimientos) que permiten la interacción entre diferentes componentes del software para obtener acceso a diferentes servicios.

APLICACIÓN ON LINE: programa creado por un desarrollador de software para que pueda accederse a través de Internet en cualquier momento y desde cualquier lugar.

BUG: es una falla en un programa informático que provoca un error o un mal funcionamiento.

CICLO DE VIDA DEL DESARROLLO DE SOFTWARE (SDLC): es todo el proceso necesario para llevar a cabo un desarrollo de software desde su planificación hasta su entrega como producto terminado.

CÓDIGO FUENTE: es el conjunto de archivos con instrucciones, realizadas en un lenguaje de programación, que conforman un programa informático y le permiten que ejecute las tareas para las que fue desarrollado.

COOKIE: es un trozo de información que almacena el navegador web del cliente y que permite conocer cierta información del sitio web visitado, de forma que se ajusten automáticamente ciertas preferencias y configuraciones.

CORTAFUEGOS (FIREWALL): es un sistema de hardware o software que sirve para proteger un sistema informático de ataques, analizando el tráfico de datos en función de ciertas reglas configurables.

DOM (DOCUMENT OBJECT MODEL): es la estructura estandarizada de objetos que genera el navegador cuando carga una página, y que puede ser accedida por el aplicativo para modificar el contenido, la estructura o el estilo del documento.

DOS (DENIAL OF SERVICE): Es un ataque cibernético que consiste en sobrecargar un sistema para que este deje de funcionar, o funcione muy lentamente.

ENCRIPTACIÓN: es un proceso que permite volver ilegible la información con el fin de protegerla frente a terceros. Únicamente el destinatario de los datos puede desencriptarlos para poder acceder y utilizar la información transmitida.

EXPLOIT: es un programa o código que se aprovecha de una vulnerabilidad en una aplicación o sistema para realizar un ataque informático.

FRAMEWORK: es un esquema predefinido que permite organizar y estructurar el código fuente y las instrucciones en el desarrollo y/o la implementación de una aplicación informática.

FREELANCER: persona que se dedica a hacer desarrollos de software por su propia cuenta sin estar vinculado laboralmente con ninguna empresa.

HTTP (HIPERTEXT TRANSFER PROTOCOL): es el protocolo utilizado por los servidores web y por los navegadores cliente para transmitir la información a través de Internet.

JSON (JAVASCRIPT OBJECT NOTATION): es un formato basado en texto plano para el intercambio de datos, independiente del lenguaje de programación utilizado.

LDAP (LIGHTWEIGHT DIRECTORY ACCESS PROTOCOL): es un protocolo que permite acceder a las bases de datos de usuarios de una red a través del protocolo TCP/IP.

LENGUAJE DE PROGRAMACIÓN: es un modo particular de escribir las instrucciones que debe ejecutar un sistema, utilizando una sintaxis, terminología y estructura predefinidos, de forma que pueda ser interpretado correctamente por dicho sistema. Algunos ejemplos de lenguajes de programación para aplicativos on line son: HTML, PHP, CSS, .Net, Java, JavaScript, Python, Perl, Ruby, etc.

LOG: es un conjunto de archivos de texto plano donde se registran cronológicamente los eventos que ocurren al utilizar un sistema informático, incluyendo las fallas o errores detectados.

MALWARE: es un tipo de código informático cuya función es dañar un sistema o causar un funcionamiento incorrecto. Incluye los virus, gusanos, troyanos, espías, rootkits, etc.

PERFIL DE USUARIO: es el conjunto de permisos y acciones autorizadas que un usuario puede realizar dentro de una aplicación, como por ejemplo los módulos y formularios a los que puede acceder y las acciones que poder llevar a cabo en cada uno de ellos.

PRUEBAS DE PENETRACIÓN (PENTESTING): son pruebas que se realizan con herramientas especializadas para detectar vulnerabilidades en un sistema informático.

PUERTO DE COMUNICACIÓN: es un punto de acceso a un sistema informático que es utilizado por un protocolo para intercambiar información con el exterior. Cada protocolo

tiene un puerto definido, de forma que el sistema puede saber qué protocolo se está utilizando en cada uno de los puertos habilitados para la comunicación.

SALIDA EN VIVO: es el momento en el cual un software o una parte de él es puesto en producción, es decir, se encuentra listo para su uso.

SAST (STATIC APPLICATION SECURITY TESTING): son las herramientas de software diseñadas para analizar código fuente o código compilado, y detectar fallas de seguridad en el mismo.

SEGURIDAD INFORMÁTICA: es la disciplina que establece los procedimientos y herramientas necesarios para proteger la confidencialidad, integridad y disponibilidad de la información.

SERIALIZACIÓN DE DATOS: es un proceso mediante el cual se transforman objetos de un programa en ejecución, en flujos de bytes que pueden ser almacenados en dispositivos, en bases de datos o ser transmitidos por la red, y después ser reconstruidos.

SERVIDOR WEB: es un programa informático que aloja el código fuente de las aplicaciones web y responde las solicitudes de los usuarios para acceder a estas aplicaciones, utilizando el protocolo HTTP.

SQL (STRUCTURED QUERY LANGUAGE): es el lenguaje utilizado por los motores de bases de datos para realizar operaciones sobre los datos o sobre su estructura.

SSL (SECURE SOCKET LAYER): es un protocolo que permite que la información que viaja a través de un medio o canal informático vaya encriptada y de esta manera no pueda ser interceptada.

TOKEN: es un mecanismo de autenticación que consiste en una cadena de caracteres y permite al servidor web identificar al usuario y otorgarle permisos de navegación en las páginas de una aplicación on line de acuerdo a su perfil de usuario, después que el usuario se ha autenticado exitosamente en la aplicación.

URL (UNIFORM RESOURCE LOCATOR): es un formato estándar para nombrar recursos en Internet, de forma que puedan identificarse unívocamente

VULNERABILIDAD: fallas o errores no intencionales que puede presentar un sistema informático por defectos en su diseño o en su implementación y configuración.

WAF (WEB APPLICATION FIREWALL): es un cortafuegos especializado en proteger los aplicativos alojados en un servidor web.

XML (EXTENSIBLE MARKUP LANGUAGE): es una especificación para diseñar lenguajes de marcado, que permite definir etiquetas personalizadas para descripción y organización de datos, permitiendo la compatibilidad entre diferentes sistemas.

XSD (XML SCHEMA DEFINITION): es un mecanismo para comprobar la validez de un documento XML, verificando su estructura, sus elementos, su tipo de datos, sus atributos, etc.

RESUMEN

Para desarrollar una aplicación on line que incluya un esquema de seguridad informática, debe aplicarse una metodología que permita implementar dicha seguridad desde la primera etapa del ciclo de vida del desarrollo de software (SDLC) y siga el proceso a lo largo de todo el ciclo hasta el momento del despliegue o salida en vivo. Entre más pronto se implemente el esquema de seguridad, menos costoso va a ser el software, ya que una modificación o cambio de estructura en etapas tardías del ciclo debido a una falla de seguridad implicaría una reingeniería del desarrollo y por tanto más tiempo de desarrollo y mayores costos.

En cada una de las etapas es posible identificar los mecanismos de implementación de la seguridad y así poder desplegarlos oportunamente.

En Colombia no existe una metodología para la incorporación de seguridad en el desarrollo de software, por eso es necesario analizar las metodologías existentes para que las empresas de desarrollo de software conozcan las alternativas existentes y puedan seleccionar la que mejor se acomode a su esquema de desarrollo y la implementen de forma que en el futuro sus desarrollos sean más seguros.

INTRODUCCIÓN

El auge del Internet y de las comunicaciones globales ha impulsado el desarrollo de aplicativos empresariales y comerciales que puedan accederse desde cualquier lugar y en cualquier momento.

Esta posibilidad de acceso sin límites supone un nuevo desafío en cuanto a seguridad de la información se refiere. Pues así como los clientes o los empleados de una empresa pueden acceder a este software, así mismo los ciberdelincuentes o cualquier persona con malas intenciones podrían intentar también acceder y causar algún tipo de daño.

Esto es un aspecto determinante a la hora de desarrollar o modificar un software de forma que implemente un esquema de seguridad informática, pues adelantar esta labor en un aplicativo ya desarrollado suele ser más complicado que realizar la implementación a lo largo de todo el ciclo de desarrollo del software, desde su planificación hasta la entrega del producto final.

Por eso, con este trabajo se pretende dar a conocer las principales metodologías existentes para la implementación de un esquema de seguridad informática en el proceso de desarrollo de los aplicativos on line, de forma que el desarrollador de software pueda comprender el ámbito de aplicación de cada metodología y de esta forma escoger la que mejor se adapte a su esquema de trabajo.

1. PLANTEAMIENTO DEL PROBLEMA

La industria del desarrollo de software para aplicaciones on line se encuentra en pleno auge en Colombia. Cada día son más las empresas nacientes que se dedican a esta actividad. Sin embargo, son pocas las que realmente dedican tiempo y esfuerzo a establecer o adoptar protocolos que les permitan crear aplicaciones que cumplan con estándares de seguridad que las protejan de manera más confiable y eficaz de los ataques que puedan ser ejecutados por hackers o personas inescrupulosas que simplemente desean un beneficio de cualquier índole: económico, reconocimiento social, venganza o cualquier otro.

Hoy en día no es necesario ser un experto en informática y programación para poder perpetrar un ataque a un sistema informático. En Internet se consiguen herramientas muy económicas, e incluso gratuitas, que permiten vulnerar la seguridad de los aplicativos.

La interconexión de los sistemas permite que la información se encuentre disponible desde cualquier parte. Las redes empresariales pueden accederse remotamente con solo digitar un usuario y una contraseña. Todo esto facilita la labor de los delincuentes informáticos que, aprovechando las aplicaciones on line, intentarán acceder a información confidencial. Este es el nuevo reto para las empresas desarrolladoras de software: proteger sus productos desde la concepción y desarrollo de los mismos, para así enfrentar de mejor manera los desafíos que se presentan.

2. JUSTIFICACIÓN

La seguridad informática es un tema que se está volviendo cada vez más importante y que está adquiriendo mayor relevancia debido al incremento de los ataques cibernéticos alrededor del mundo.

En el caso de los aplicativos on line, este tema se vuelve crucial, ya que cualquier persona, desde cualquier lugar del planeta, puede intentar acceder a la información de uno de estos aplicativos y generar un ataque que comprometa la información de la empresa y deje en entredicho la credibilidad de la misma.

Por esta razón es importante analizar las metodologías que pueden ser usadas por las empresas desarrolladoras de software on line, para que, durante el proceso de desarrollo, puedan aplicar el esquema de seguridad propuesto en la metodología y así puedan garantizar un software más seguro a sus clientes.

Al lograr esto, la empresa desarrolladora tendrá una ventaja competitiva frente a las demás empresas, pues podrá ofrecer un producto de mejor calidad ya que cumplirá con los estándares más altos de seguridad implementados desde el proceso mismo del desarrollo.

Siendo Colombia un país con altísima producción de software, este trabajo adquiere mucha importancia pues el campo de acción es muy amplio y muchas de estas empresas podrán aplicar estas metodologías en el desarrollo de sus productos, impactando positivamente a la sociedad y contribuyendo de igual forma al campo de la seguridad informática, pues un software donde se integre la seguridad desde el principio genera confianza en el consumidor y redunda en mayores ingresos para la empresa desarrolladora, generando empleo y credibilidad en los clientes.

3. OBJETIVOS

3.1 OBJETIVO GENERAL

Realizar un análisis de las metodologías de desarrollo de software seguro on line.

3.2 OBJETIVOS ESPECÍFICOS

- Determinar los tipos de vulnerabilidades y amenazas a los que están expuestas las aplicaciones on line.
- Identificar y comparar los estándares existentes en cuanto a seguridad en el desarrollo de aplicativos se refiere.
- Determinar las diferentes etapas existentes en el proceso de desarrollo de software y establecer los esquemas de seguridad aplicables en cada una de las etapas.
- Determinar el procedimiento a seguir para realizar los diferentes test de seguridad a lo largo del desarrollo del software.

4. VULNERABILIDADES DE LOS APLICATIVOS ON LINE

La seguridad del software constituye una propiedad emergente determinada por la interacción de múltiples factores a lo largo del proceso de desarrollo, desde su concepción hasta su muerte. En este sentido, es importante diferenciar la seguridad del código desarrollado con las características de implementación o instalación, como el caso del protocolo SSL o la utilización de *firewalls*[1].

En el caso de los aplicativos on line, un atacante intentará no ser descubierto por las herramientas que puedan estar protegiendo el aplicativo, tales como un *firewall* o un WAF (*web application firewall*). Por esta razón existen herramientas que le permiten al atacante conocer si un sitio web está correctamente configurado, sin necesidad de realizar un escaneo de puertos que lo pueda delatar. Un ejemplo es Shodan, que es un buscador que permite almacenar la información que obtiene de un dispositivo expuesto a Internet (*routers*, cámaras IP, etc.) tales como: claves, direcciones IP, puertos abiertos, servicios y versiones, certificados SSL, etc.[2]

El atacante también podría acceder a los metadatos que se incluyen en los documentos publicados en Internet. Con ellos es posible conocer nombres de usuario, equipos, sistemas operativos, versiones de software, etc. que utiliza la organización, y llegar incluso a realizar un mapa interno que incluya servidores, impresoras, equipos cliente, usuarios, etc. Esto se puede prevenir con herramientas como FOCA, que pueden verificar la información sensible que se está publicando en Internet sin ser consciente de ello[3].

De igual forma, el atacante podría intentar un ataque directo al aplicativo on line, ingresando código malicioso en los formularios web, de forma que consiga manipular el resultado entregado por el servidor y así redireccionar al usuario a un sitio malicioso o incluso acceder a la base de datos y manipular o extraer información de la misma.

En la siguiente imagen se observan los resultados del estudio hecho por la ENISA (*European Union Agency for Network and Information Security*) donde se establece que los ataques basados en web y los ataques a aplicaciones web ocuparon el segundo y tercer lugar de la lista respectivamente durante los años 2017 y 2018[4]:

---

[1] GIUSTO, Denise. 10 consejos para el desarrollo seguro de aplicaciones. Welivesecurity en español [en línea]. Disponible en: https://www.welivesecurity.com/la-es/2015/03/12/10-consejos-desarrollo-seguro-de-aplicaciones/

[2] MONTES, María José. Las principales vulnerabilidades web. Hacking Ético [en línea]. Disponible en: https://hacking-etico.com/2017/04/04/las-principales-vulnerabilidades-web/

[3] Ibíd.

[4] ENISA. ENISA Threat Landscape Report 2018. ENISA [en línea]. Disponible en: https://www.enisa.europa.eu/publications/enisa-threat-landscape-report-2018

Figura 1. Top 15 de amenazas informáticas. Comparativo 2017-2018

Top Threats 2017	Assessed Trends 2017	Top Threats 2018	Assessed Trends 2018	Change in ranking
1. Malware	➲	1. Malware	➲	→
2. Web Based Attacks	↑	2. Web Based Attacks	↑	→
3. Web Application Attacks	↑	3. Web Application Attacks	➲	→
4. Phishing	↑	4. Phishing	↑	→
5. Spam	↑	5. Denial of Service	↑	↑
6. Denial of Service	↑	6. Spam	➲	↓
7. Ransomware	↑	7. Botnets	↑	↑
8. Botnets	↑	8. Data Breaches	↑	↑
9. Insider threat	➲	9. Insider Threat	↓	→
10. Physical manipulation/ damage/ theft/loss	➲	10. Physical manipulation/ damage/ theft/loss	➲	→
11. Data Breaches	↑	11. Information Leakage	↑	↑
12. Identity Theft	↑	12. Identity Theft	↑	→
13. Information Leakage	↑	13. Cryptojacking	↑	NEW
14. Exploit Kits	↓	14. Ransomware	↓	↓
15. Cyber Espionage	↑	15. Cyber Espionage	↓	→

Legend: Trends: ↓ Declining, ➲ Stable, ↑ Increasing
Ranking: ↑ Going up, → Same, ↓ Going down

Fuente: https://www.enisa.europa.eu/publications/enisa-threat-landscape-report-2018

4.1 VULNERABILIDADES DEL ENTORNO

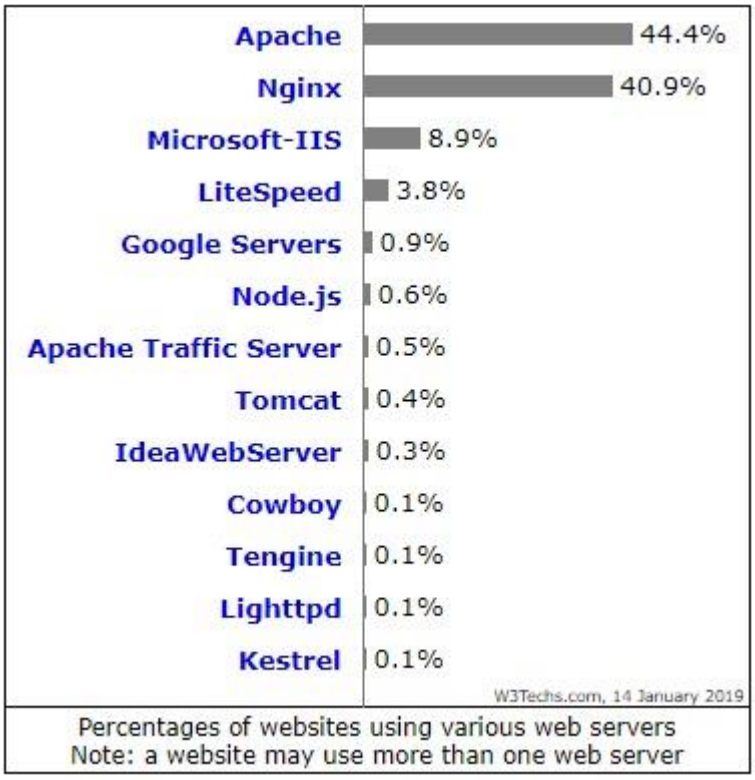
Son aquellas que no se deben al aplicativo on line en sí, pero que si pueden afectar su funcionamiento.

4.1.1 Configuración débil. Ocurre cuando un sistema informático, sistema operativo o aplicación se configura sin tener el suficiente conocimiento, tiempo o experiencia en el mismo, o cuando se dejan establecidas las opciones por defecto. Cuando no se modifica

la configuración por defecto, las vulnerabilidades de esa configuración probablemente ya sean conocidas y el ataque será efectivo. Es necesario siempre modificar estas configuraciones para hacer los servicios más robustos y seguros.

Dado que los aplicativos on line se deben instalar en un servidor web para que puedan funcionar y estar disponibles para los usuarios en Internet, es imprescindible establecer la configuración adecuada para este servidor. Existen diferentes tipos de servidores web: Apache, Microsoft IIS, Sun Java, Nginx, Google Web Server (GWS) y otros[5]. Cada uno tiene uno o más archivos u opciones de configuración, y es responsabilidad del administrador del servidor configurar el mismo para mejorar su seguridad y aprovechar al máximo el hardware que tiene a disposición. En la siguiente imagen se puede observar la cuota de mercado de los principales servidores web disponibles[6].

Figura 2. Cuota de mercado de los diferentes servidores web.



Fuente: https://maslinux.es/nginx-vs-apache-cual-es-el-mejor-servidor/

[5] CARDADOR CABELLO, Antonio Luis. Implantación de aplicaciones web en entornos internet, intranet y extranet. 1 ed. Málaga. IC Editorial, 2014. p. 70-75.
[6] BORAL, Sayak. Nginx vs Apache: ¿Cuál es el mejor servidor? MasGNULinux [en línea]. Disponible en: https://maslinux.es/nginx-vs-apache-cual-es-el-mejor-servidor/

Como ejemplo, el servidor web Apache tiene un archivo de configuración denominado *httpd.conf*. Este archivo viene en texto plano por lo que podrá editarse con cualquier editor de textos. Tiene tres partes principales: configuración global, configuración principal y *hosts* virtuales. Dentro de la configuración global se puede definir la ruta del directorio donde se alojan las páginas web, el tiempo de vida de las peticiones HTTP, el puerto de escucha de las peticiones, y los módulos adicionales a utilizar. Dentro de la configuración principal se puede establecer el directorio del aplicativo web, el archivo de registro de errores, el tipo de errores que mostrará el aplicativo, el tipo de archivos índice que ejecutará el servidor, las acciones específicas dependiendo del tipo de navegador del cliente, y otras opciones de seguridad especiales[7]. Si no se configura este archivo y se deja tal cual es instalado, un atacante podrá conocer la versión de servidor web que se está ejecutando y de esta manera explotar sus vulnerabilidades conocidas. En la siguiente imagen se puede observar un archivo httpd.conf de ejemplo.

Figura 3. Ejemplo de archivo httpd.conf de configuración de Apache

Fuente: http://www.thaiall.com/blog/tag/apache/

En cuanto al sistema operativo del servidor web, es recomendable ajustar la configuración del mismo de forma que se proteja frente a accesos no autorizados y ataques de malware. Entre las recomendaciones principales, se encuentran:

- Limitar el número de usuarios que tienen acceso al servidor. Deshabilitar los usuarios que se instalan por defecto y que no se utilizan (p. ej. usuario invitado).

- Asignar una contraseña robusta a todos los usuarios que tengan acceso al servidor.

- Utilizar una sola cuenta de administrador y limitar su uso.

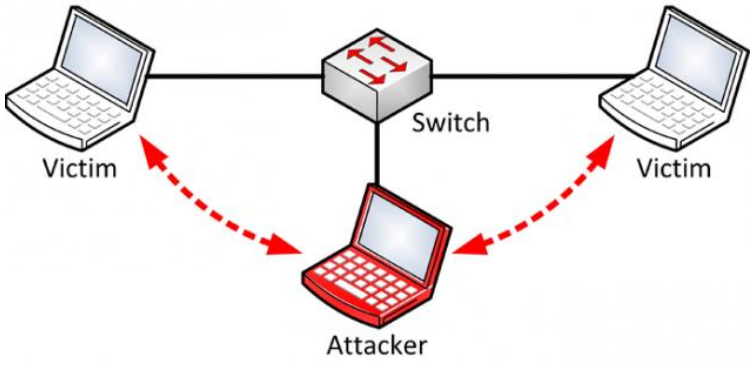
[7] CARDADOR CABELLO. Op. cit., p. 81.

- Establecer políticas de seguridad para el servidor y para la red en que se encuentra.
- Deshabilitar los servicios y aplicaciones que no se requieran.
- Bloquear los puertos de acceso que no se usan.
- Instalar herramientas de protección como *firewall* y antivirus, configurarlos correctamente y verificar que se actualicen diariamente.
- Realizar el seguimiento y control a los procesos realizados en el servidor a través de los *logs* que se generan[8].

4.1.2 Comunicación insegura. Una parte importante de los aplicativos on line es la comunicación que se establece entre el cliente y el servidor web. Dado que esta comunicación se lleva a cabo generalmente a través de Internet, es importante proteger los datos que se transmiten y reciben, sobre todo cuando se envían credenciales de autenticación o datos privados o relevantes a través de formularios web.

Un atacante cualquiera puede interceptar fácilmente la comunicación establecida y acceder a la información con el fin de robarla, modificarla o bloquearla; esto se debe a que el protocolo HTTP transmite la información tal cual se genera, es decir, sin ningún tipo de encriptación. A este tipo de ataque se le denomina *Man in the Middle* (MitM), y se puede observar su implementación en la siguiente imagen:

Figura 4. Esquema de un ataque tipo Man in the Middle



Fuente: https://www.redeszone.net/2016/12/06/mitmap-programa-uno-realizar-ataques-man-in-the-middle/

[8] Ibíd., p. 83-84.

Existen herramientas de software especialmente diseñadas para esta labor, como por ejemplo *Wireshark*, *mitmAP*, *SSLstrip* o *Driftnet*, que ejecutan diferentes tareas para vulnerar la comunicación y lograr que el ataque sea efectivo[9].

Para enfrentar este tipo de ataques existen una serie de protocolos de seguridad que permiten encriptar la información mientras viaja por Internet. Los más conocidos son SSL (*Secure Socket Layer*) y TLS (*Transport Layer Security*). Estos utilizan un certificado digital emitido por una autoridad certificadora (CA) que garantiza la autenticidad del sitio web. El usuario puede saber si la comunicación es segura al observar en la URL del sitio si se está utilizando el protocolo HTTPS en lugar de HTTP[10]. La siguiente imagen muestra un ejemplo de utilización del protocolo HTTPS en la página web de la UNAD.

Figura 5. Utilización del protocolo HTTPS en una página web

Fuente: elaboración propia

Algunas versiones antiguas del protocolo como SSLv2 y SSLv3 ya son inseguras, y si se utilizan darán una falsa sensación de seguridad, pues el cliente verá que existe un certificado SSL y por tanto confiará en el aplicativo e ingresará al mismo, cuando el atacante podría estar descifrando la comunicación[11].

4.1.3 Software desactualizado. Mantener los aplicativos y el sistema operativo sin actualizar y sin los parches de seguridad que liberan los fabricantes significa exponer los mismos a los ataques que se aprovechan de las vulnerabilidades detectadas (*exploits*).

Cada fabricante de hardware y software (sistemas operativos y aplicaciones) suelen lanzar parches de forma periódica para corregir ciertas vulnerabilidades a medida que van siendo detectadas. Esto garantiza que el aplicativo o el hardware han incrementado sus

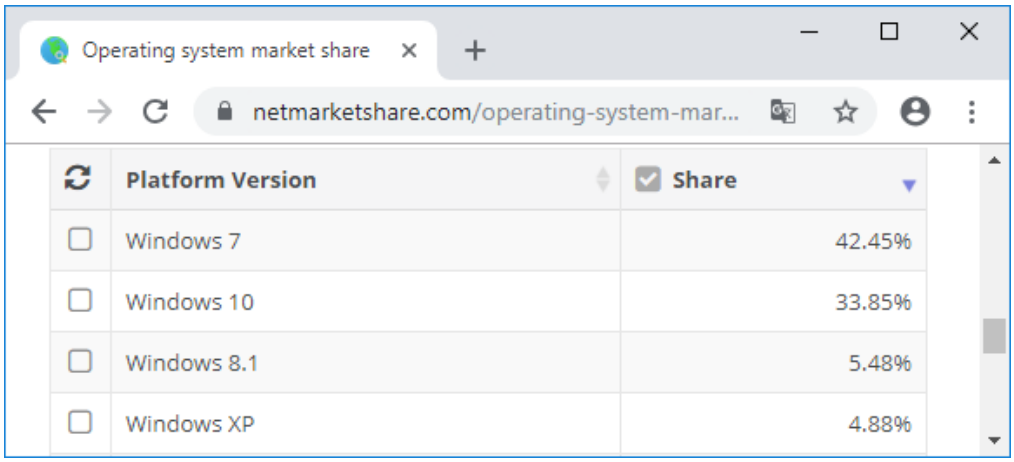
[9] DE LUZ, Sergio. mitmAP: Un programa todo en uno para realizar ataques Man In The Middle. Redes Zone [en línea]. Disponible en: https://www.redeszone.net/2016/12/06/mitmap-programa-uno-realizar-ataques-man-in-the-middle/
[10] CASTRO, Luis. ¿Qué es SSL o TLS?. About Español [en línea]. Disponible en: https://www.aboutespanol.com/que-es-ssl-o-tls-157625
[11] MONTES. Op. cit.

niveles de seguridad. Sin embargo, ocurre con frecuencia que estas actualizaciones no se realizan de forma automática, sino que requieren de la intervención del usuario para ser descargadas e instaladas. Por este motivo muchos sistemas permanecen desactualizados y vulnerables durante bastante tiempo, incluso años.

Así mismo, se encuentran instalados muchos equipos y software obsoletos alrededor del mundo, que ya no reciben actualizaciones de seguridad por parte del fabricante, y que por tanto están completamente expuestos a ataques. Esto sucede por el alto costo de adquisición y licenciamiento que tienen los nuevos sistemas, razón por la cual los usuarios optan por no comprar las últimas versiones. Como ejemplo se puede citar el caso del sistema operativo Windows XP, el cual fue lanzado en el año 2002 y aún se sigue utilizando en el 4,88% de los computadores, a pesar de que su soporte extendido finalizó el 8 de abril de 2014[12].

A continuación se puede observar la cuota de mercado de las diferentes versiones del sistema operativo Windows para el segundo semestre de 2018.

Figura 6. Cuota de mercado de las diferentes versiones de Windows



Fuente: https://netmarketshare.com/operating-system-market-share.aspx

4.1.4 Almacenamiento inseguro de los datos. Todos los aplicativos on line actuales se conectan a una base de datos para almacenar la información que recolectan y entregan a los usuarios. Estas bases de datos se gestionan mediante un motor de base de datos como MySQL, PostgreSQL, Oracle, MS-SQL Server, MongoDB, etc. Cada uno de estos motores se puede configurar para maximizar el rendimiento y la seguridad conforme a los

[12] MICROSOFT CORPORATION. Finalizó el soporte para Windows XP. Microsoft [en línea]. Disponible en: https://www.microsoft.com/es-xl/windowsforbusiness/end-of-xp-support

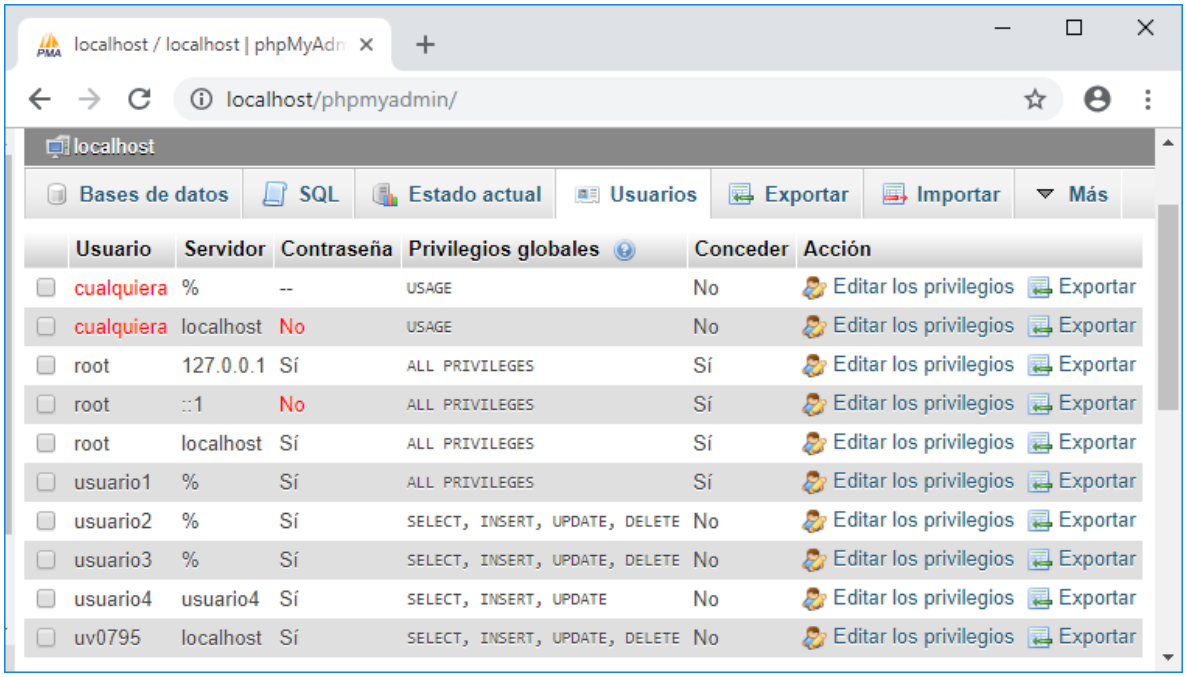
recursos de hardware de que dispone y las necesidades de los usuarios. Sin embargo su configuración por defecto es muy básica y por eso siempre debe mejorarse. Por ejemplo, algunas bases de datos permiten crear un superusuario sin contraseña, lo que permitiría a cualquier atacante ingresar a la misma sin ninguna restricción.

Por otra parte, también se debe proteger la información sensible almacenada en la base de datos en caso que un atacante pueda acceder a ella. Para esto es importante encriptar dicha información con mecanismos como AES o MD5, los cuales ya vienen implementados dentro de los motores de bases de datos y simplemente deben activarse.

Por último, es importante limitar al máximo los permisos del usuario que se conecta a la base de datos desde el aplicativo on line, pues en caso que el atacante logre acceder a la base de datos desde el aplicativo, no podrá llevar a cabo sino las acciones permitidas al usuario que establece la conexión.

La siguiente imagen muestra un ejemplo de los usuarios creados en una base de datos MySQL y los privilegios que ostentan cada uno de estos usuarios. Se observa que se han creado usuarios con privilegios limitados y otros que se pueden conectar únicamente de forma local o desde un determinado servidor.

Figura 7. Configuración de usuarios en MySQL



Fuente: elaboración propia

4.2 VULNERABILIDADES DEL APLICATIVO

Son las que se relacionan directamente con el desarrollo del software, el código fuente o la ejecución del mismo. El esquema de seguridad informática que se implemente debe atacar estas vulnerabilidades y corregirlas para lograr su cometido principal. Existe una entidad sin ánimo de lucro que se encarga de difundir periódicamente el top 10 de los riesgos más críticos en aplicaciones web. Esta entidad es el OWASP (*Open Web Application Security Project*), una fundación cuya misión es, desde el 2001, el "permitir que las organizaciones desarrollen, adquieran y mantengan aplicaciones y APIs en las que se pueda confiar"[13].

4.2.1 *Cross site scripting* (XSS). La ejecución de comandos en sitios cruzados ocurre cuando se cargan datos no validados en un navegador, permitiendo ejecutar comandos que secuestren sesiones, modifiquen los sitios web o redireccionen al usuario a un sitio malicioso. Según la OWASP, es la segunda vulnerabilidad más frecuente y se encuentra en el 67% de todas las aplicaciones desarrolladas. Además, existen herramientas automáticas que pueden detectar y explotar la vulnerabilidad, y también se consiguen kits de explotación gratuitos[14]. La siguiente imagen muestra un ejemplo de ataque XSS descrito en el documento del OWASP Top 10.

Figura 8. Ejemplo de ataque XSS

Ejemplos de escenarios de ataque

Escenario 1: la aplicación utiliza datos no confiables en la construcción del código HTML sin validarlos o codificarlos:

```
(String) page += "<input name='creditcard' type='TEXT' value='" +
request.getParameter("CC") + "'>";
```

El atacante modifica el parámetro "CC" en el navegador por:

```
'><script>document.location='http://www.attacker.com/cgi-
bin/cookie.cgi?foo='+document.cookie</script>'
```

Este ataque causa que el identificador de sesión de la víctima sea enviado al sitio web del atacante, permitiéndole secuestrar la sesión actual del usuario.

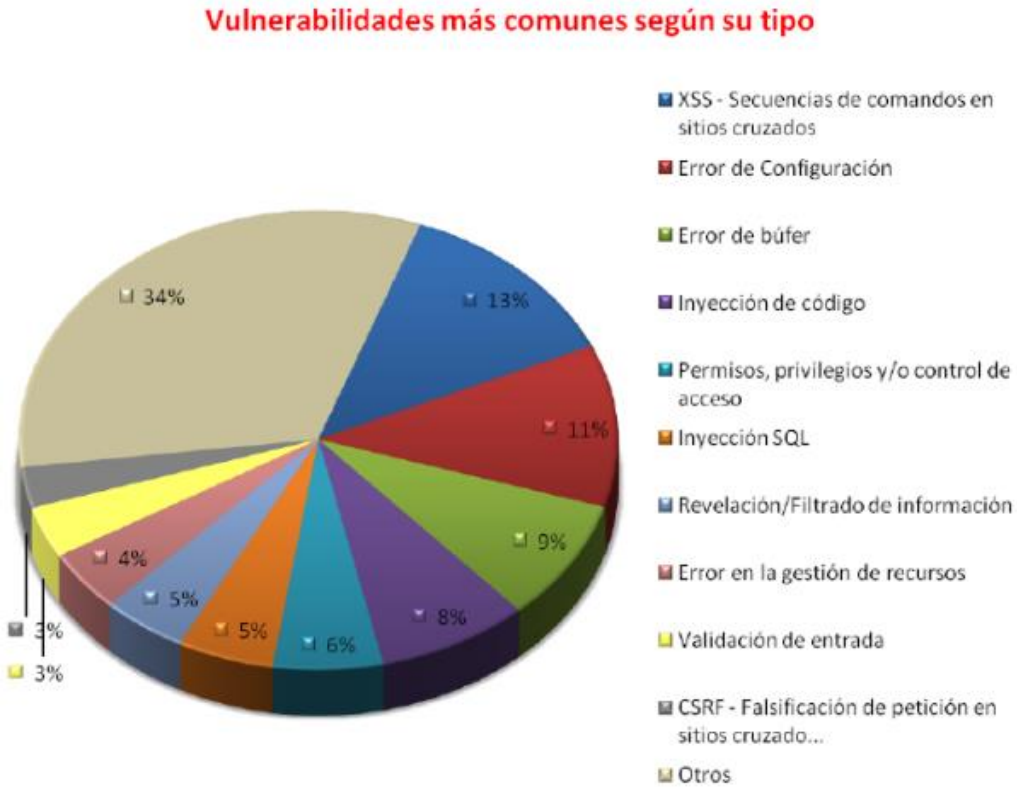
Nota: los atacantes también pueden utilizar XSS para anular cualquier defensa contra Falsificación de Peticiones en Sitios Cruzados (CSRF) que la aplicación pueda utilizar.

Fuente: The OWASP Foundation. OWASP Top 10 – 2017 Los diez riesgos más críticos en Aplicaciones Web

[13] The OWASP Foundation. OWASP Top 10–2017 Los diez riesgos más críticos en aplicaciones web. 2017. p. 1
[14] Ibíd., p. 13.

La siguiente gráfica muestra cómo el XSS es una de las vulnerabilidades más comunes en los entornos web.

Figura 9. Vulnerabilidades más comunes según su tipo



Fuente: https://www.welivesecurity.com/la-es/2012/07/31/vulnerabilidades-estadisticas-e-impacto/#single-post-fancybox-2

Existen tres formas de XSS[15]:

- XSS reflejado. La aplicación utiliza datos proporcionados por el usuario que no han sido validados y que el atacante codifica como HTML o JavaScript, permitiendo ejecutar comandos en el navegador de la víctima.

- XSS almacenado. La aplicación almacena los datos suministrados por el usuario que no han sido validados, y que son posteriormente utilizados por otro usuario sin sospechar que son fraudulentos y que pueden generar un ataque.

- XSS basado en DOM. *Frameworks*, aplicaciones o APIs que incluyen datos dinámicos que pueden ser controlados por un delincuente.

[15] Ibíd., p. 13.

Para prevenir ataques XSS se pueden llevar a cabo las siguientes tareas[16]:

- Uso de *frameworks* seguros. Estos ya incluyen código para prevenir ataques XSS. Por ejemplo: Ruby o React JS.

- Codificar los datos no confiables HTTP de los campos de formulario.

- Aplicar codificación sensitiva al contexto en el navegador cliente.

- Crear y aplicar una política de seguridad de contenido que permita mitigar XSS.

4.2.2 *Cross site request/reference forgery* (CSRF). La falsificación de solicitudes entre sitios es una evolución del XSS, donde se engaña al usuario para que realice peticiones no autorizadas en la aplicación en la que se encuentra autenticado. El atacante obtiene la identidad y los privilegios de la víctima, lo que le permite realizar acciones en nombre de ella, como por ejemplo cambiar de estado en el servidor, cambiar la contraseña de acceso o el correo electrónico, o realizar compras en línea. Si el sitio vulnerable permite almacenar el ataque, la gravedad del mismo se amplía porque es muy probable que la víctima acceda a la página que contiene el ataque, pues ya se encuentra autenticada en el sitio[17].

Otros nombres que puede recibir este tipo de ataque son: XSRF, Sea Surf, Session Ridding, Hostile linking o One-click[18].

Ejemplo de ataque: una persona desea hacer una transferencia bancaria. La página del banco hace la entrega de los datos por el método GET de esta forma:

GET http://mibanco.com/transferencia?cta=JUANPEREZ&cantidad=100000

El atacante utiliza ingeniería social para que la víctima cargue una URL maliciosa mientras hace la transacción en el banco. Puede hacerlo enviando un correo con el contenido HTML malicioso escondido en un enlace o una imagen falsa, o colocando la URL dentro de una página que la víctima pueda visitar sin sospechar nada. El enlace con la URL falsa podría ser así:

[16] Ibíd., p. 13.
[17] The OWASP Foundation. Cross-Site Request Forgery (CSRF). OWASP [en línea]. Disponible en: https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)
[18] Ibíd.

* !Vea esta imagen! *

O, utilizando una imagen escondida de tamaño 0x0:

<img src="http://mibanco.com/transferencia?cta=ELATACANTE&cantidad= 10000000" width = "0" height = "0" border = "0">

El mismo tipo de ataque se puede realizar a través de los métodos POST o PUT.

Afortunadamente, los navegadores modernos ya incluyen protecciones contra este tipo de ataques, por lo que su efectividad ha disminuido considerablemente, hasta el punto que ya no aparece en el Top 10 de riesgos críticos de la OWASP en su versión 2017. Sin embargo, aún se pueden realizar ciertas tareas para prevenir estos ataques:

- Agregar un id adicional a los formularios, aparte del id de sesión. Este id es conocido como "clave de formulario". Los *frameworks* modernos ya incluyen este tipo de protección.

- Agregar un hash a todos los formularios web con información de la sesión y una clave secreta, de forma que pueda validarse el origen del formulario.

- Hacer una verificación del encabezado HTTP para comprobar que la solicitud proviene del sitio original y no de uno falsificado.

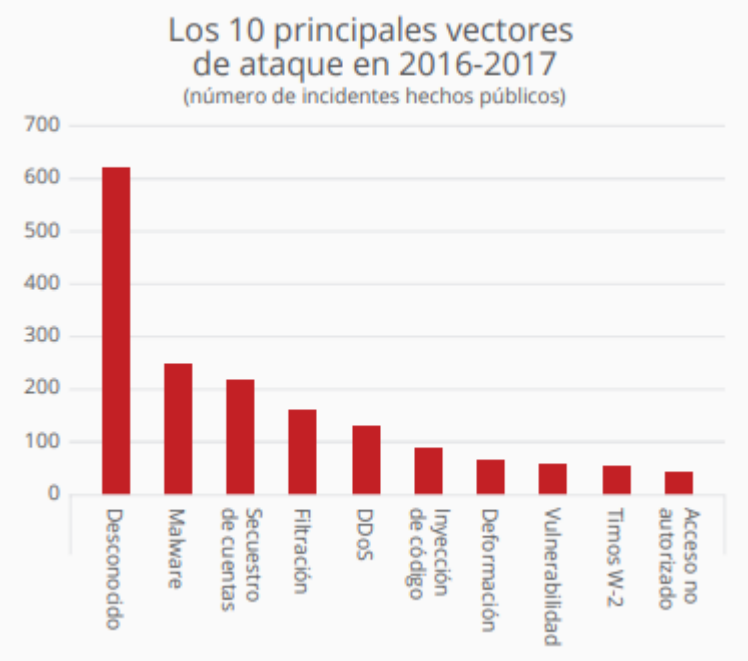
- Los usuarios deben siempre cerrar sus sesiones antes de visitar un sitio nuevo o pueden borrar las cookies del navegador antes de cerrarlo[19].

4.2.3 *SQL injection*. Esta vulnerabilidad permite al atacante acceder a la base de datos donde el aplicativo on line almacena la información recogida, y manipular, eliminar o sustraer datos, o escalar privilegios de usuario. Para lograr la inyección, se articulan consultas a la base de datos en lenguaje SQL desde los campos de ingreso de los formularios HTML y así se consigue el acceso[20].

Este tipo de ataque se ha extendido a consultas NoSQL, a comandos del sistema operativo o a comandos LDAP, y sigue siendo uno de los ataques más presentados a nivel mundial. En la siguiente gráfica se puede observar la ubicación de este ataque dentro de los 10 principales ataques de todo tipo en los años 2016 y 2017.

[19] Ibíd.
[20] MONTES. Op. cit.

Figura 10. Los 10 principales vectores de ataque en 2016-2017



Fuente: https://www.mcafee.com/enterprise/es-es/assets/reports/rp-quarterly-threats-mar-2018.pdf

En la imagen siguiente se observan ejemplos de ataques por SQL injection tomados del documento del OWASP Top 10.

Figura 11. Ejemplo de inyección SQL

Ejemplos de escenarios de ataque

Escenario #1: la aplicación utiliza datos no confiables en la construcción del siguiente comando SQL vulnerable:

```
String query = "SELECT * FROM accounts WHERE custID='" +
request.getParameter("id") + "'";
```

Escenario #2: la confianza total de una aplicación en su *framework* puede resultar en consultas que aún son vulnerables a inyección, por ejemplo, *Hibernate Query Language (HQL)*:

```
Query HQLQuery = session.createQuery("FROM accounts WHERE
custID='" + request.getParameter("id") + "'");
```

En ambos casos, al atacante puede modificar el parámetro *"id"* en su navegador para enviar: ' or '1'='1. Por ejemplo:

```
http://example.com/app/accountView?id=' or '1'='1
```

Esto cambia el significado de ambas consultas, devolviendo todos los registros de la tabla *"accounts"*. Ataques más peligrosos podrían modificar los datos o incluso invocar procedimientos almacenados.

Fuente: The OWASP Foundation. OWASP Top 10 – 2017 Los diez riesgos más críticos en Aplicaciones Web

Estos son algunos consejos para prevenir inyecciones SQL[21]:

- Separar los datos de los comandos y de las consultas.
- Utilizar APIs seguras, evitando el uso de intérpretes y parametrizando las interfaces y/o los procedimientos almacenados cuando éstos se utilicen.
- Realizar la validación de los datos de entrada desde el servidor.
- Escapar los caracteres especiales utilizando las funciones del intérprete específicas para esta labor.
- Utilizar los comandos para limitar el número de registros obtenido en la consulta (por ejemplo el comando LIMIT).

4.2.4 Pérdida de autenticación. Se realiza una inadecuada implementación de la autenticación y la gestión de sesiones, de forma que los atacantes pueden comprometer los *token* de sesión, los usuarios y/o sus contraseñas, o suplantar a otros usuarios. Los atacantes pueden utilizar listas de usuarios y contraseñas conocidos para intentar ingresar a la aplicación, o pueden realizar ataques de fuerza bruta para descifrar la contraseña, sobre todo si se utilizan contraseñas débiles o muy conocidas[22].

¿Cómo se previene[23]?

- Utilizar autenticación de múltiple factor, evitando así ataques automatizados.
- No utilizar credenciales por defecto.
- Verificar la contraseña del usuario contra la lista del top 10.000 de peores contraseñas para evitar contraseñas débiles.
- Establecer políticas para las contraseñas, como longitud mínima, uso de mayúsculas, minúsculas, números y caracteres especiales, y vencimiento de la misma.

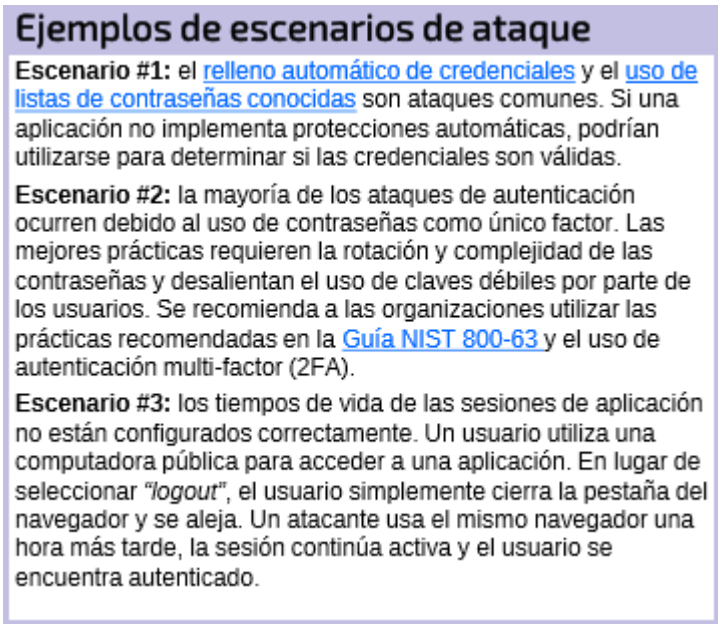
[21] The OWASP Foundation, OWASP Top 10 – 2017 Los diez riesgos más críticos en Aplicaciones Web, Op. Cit. p. 7.
[22] Ibíd., p. 6.
[23] Ibíd., p. 8.

- Limitar el número de intentos fallidos para acceder a la aplicación y llevar un registro de todos los intentos, exitosos o no.

- Nunca colocar el ID de sesión en la URL, y eliminarlo cuando se termine la sesión o después de un tiempo de inactividad determinado.

A continuación se observan, en la imagen, los diferentes escenarios de ataque que pueden presentarse como ejemplos de pérdida de autenticación.

Figura 12. Ejemplo de pérdida de autenticación



Ejemplos de escenarios de ataque

Escenario #1: el relleno automático de credenciales y el uso de listas de contraseñas conocidas son ataques comunes. Si una aplicación no implementa protecciones automáticas, podrían utilizarse para determinar si las credenciales son válidas.

Escenario #2: la mayoría de los ataques de autenticación ocurren debido al uso de contraseñas como único factor. Las mejores prácticas requieren la rotación y complejidad de las contraseñas y desalientan el uso de claves débiles por parte de los usuarios. Se recomienda a las organizaciones utilizar las prácticas recomendadas en la Guía NIST 800-63 y el uso de autenticación multi-factor (2FA).

Escenario #3: los tiempos de vida de las sesiones de aplicación no están configurados correctamente. Un usuario utiliza una computadora pública para acceder a una aplicación. En lugar de seleccionar *"logout"*, el usuario simplemente cierra la pestaña del navegador y se aleja. Un atacante usa el mismo navegador una hora más tarde, la sesión continúa activa y el usuario se encuentra autenticado.

Fuente: The OWASP Foundation. OWASP Top 10 – 2017 Los diez riesgos más críticos en Aplicaciones Web

4.2.5 Entidades externas XML (XXE). Las entidades externas XML pueden ser usadas para revelar archivos internos a través de la URI o en servidores desactualizados. También se pueden utilizar para escanear puertos de red, para ejecutar código remoto o para realizar ataques de denegación del servicio (DoS). Esto sucede cuando se utilizan procesadores XML antiguos o mal configurados[24].

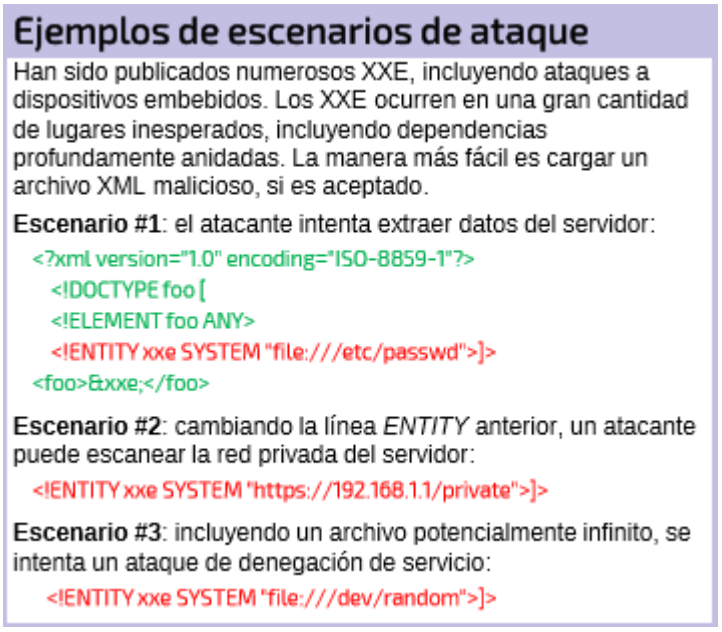
Medidas de prevención[25]:

[24] Ibíd., p. 6.
[25] Ibíd., p. 10.

- Utilizar formatos de datos simples como JSON y evitar la serialización de datos sensibles.
- Mantener actualizados los procesadores y librerías XML y utilizar validadores de dependencias.
- Deshabilitar, en lo posible, las entidades externas XML.
- Validar las entradas XML para evitar datos maliciosos en documentos, cabeceras y nodos, usando validación XSD.
- Utilizar herramientas SAST para la detección de vulnerabilidades XXE.
- Utilizar un WAF (*Web Application Firewall*) para la detección y bloqueo de XXE.

En la siguiente imagen se observan tres escenarios de ataques por XXE.

Figura 13. Ejemplo de fallos por XXE



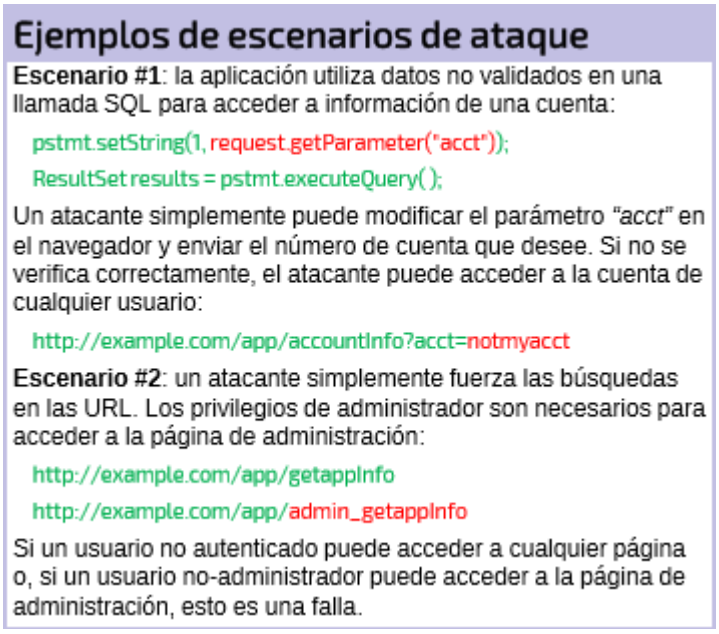
Fuente: The OWASP Foundation. OWASP Top 10 – 2017 Los diez riesgos más críticos en Aplicaciones Web

4.2.6 Pérdida de control de acceso. La definición de roles y perfiles dentro del aplicativo no se hace de manera adecuada, de forma que un atacante puede escalar privilegios para

acceder a funcionalidades o datos no autorizados, cuentas de otros usuarios, modificar, copiar o borrar datos, cambiar permisos, etc.[26]

La siguiente imagen muestra dos ejemplos de pérdida de control de acceso.

Figura 14. Ejemplo de pérdida de control de acceso



Fuente: The OWASP Foundation. OWASP Top 10 – 2017 Los diez riesgos más críticos en Aplicaciones Web

Consejos contra la pérdida de control de acceso[27]:

- Se debe establecer la política de denegación de permisos a todos los usuarios, e ir asignando permisos en la medida que se establecen los roles y perfiles a cada usuario.

- Implementar los mecanismos de control de acceso una sola vez y reutilizarlos en toda la aplicación.

- Deshabilitar el listado de archivos de directorios y subdirectorios en el servidor web y verificar que no exista código fuente en carpetas públicas.

[26] Ibíd., p. 6.
[27] Ibíd., p. 11.

- Llevar un registro de los errores en los controles de acceso e informar al administrador cuando se presenten estos errores.

- Limitar el acceso a las APIs para minimizar los ataques automatizados.

- Invalidar los *token* de acceso al finalizar la sesión del usuario.

- Incluir pruebas de control de acceso en el proceso de desarrollo del software, en las pruebas unitarias y en las pruebas integrales.

5. METODOLOGÍAS DE SEGURIDAD EN APLICACIONES

Los principales actores en la industria del software se han preocupado por desarrollar metodologías para permitir que los desarrolladores de software incluyan la seguridad informática en todas las etapas de desarrollo.

Es importante conocer las principales metodologías para así poder aplicar alguna de ellas y realizar desarrollos de software más seguros y confiables.

5.1 ESTÁNDAR DE VERIFICACIÓN DE SEGURIDAD EN APLICACIONES (ASVS)

El estándar ASVS (Application Security Verification Standard) es un marco de referencia desarrollado por la OWASP para determinar los requisitos y controles de seguridad necesarios al diseñar, desarrollar, probar y mantener aplicaciones web[28].

Dentro de este estándar se han definido tres niveles de verificación, dependiendo de las necesidades en cuanto a requerimientos de seguridad se refiere:

- Nivel 1: Oportunista. Es el nivel donde aparecen la mayoría de las aplicaciones con esquema de seguridad implementado. Una aplicación en nivel 1 puede defenderse de vulnerabilidades comunes como las que se incluyen en el OWASP Top 10. Es el mínimo nivel requerido para todas las aplicaciones.

- Nivel 2: Estándar. Es el nivel donde se encuentran las aplicaciones que se defienden de forma adecuada contra la mayoría de los riesgos presentes hoy en día. Implementa controles adecuados y efectivos, y se utilizan dentro de la aplicación. Es el nivel apropiado para aplicaciones transaccionales de negocios, que administren información de salud, que manejan datos sensibles o críticos del negocio o que procesan activos sensibles. Los ataques a este nivel son dirigidos, con técnicas y herramientas orientadas al descubrimiento y explotación de vulnerabilidades en las aplicaciones.

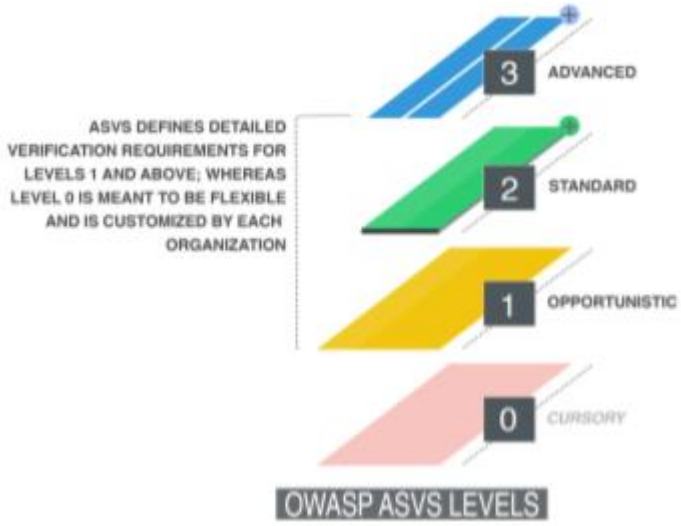
- Nivel 3: Avanzado. Es el más alto nivel dentro de ASVS. Aplica para aplicativos militares, de salud, de seguridad, infraestructura y otros que realicen funciones críticas para la entidad. La aplicación nivel 3 se defiende adecuadamente contra ataques avanzados y puede demostrar la adopción de un buen diseño de seguridad, pues requiere mayor nivel de arquitectura, codificación y pruebas que los niveles anteriores. También se caracterizan por su modularidad, con cada módulo encargado de su

[28] The OWASP Foundation, Estándar de Verificación de Seguridad en Aplicaciones 3.0.1. p. 7.

propia seguridad debidamente documentada. Los controles deben asegurar la confidencialidad (con cifrado, por ejemplo), la integridad (con validaciones transaccionales y de entrada), la disponibilidad (con balanceo de cargas), la autenticación (entre módulos y sistemas y de usuarios), y la auditoría (con bitácoras)[29].

A continuación se pueden observar los niveles del ASVS tomados del documento oficial de la OWASP.

Figura 15. Niveles del ASVS



Fuente: The OWASP Foundation, Estándar de Verificación de Seguridad en Aplicaciones 3.0.1.

Dentro del estándar se consideran 181 requisitos de verificación que debe cumplir el software para encontrarse en el nivel tres del estándar. De estos requisitos, se deben cumplir 87 para aplicar al nivel 1. Estos requisitos se encuentran enmarcados dentro de 16 módulos[30]:

- Arquitectura, diseño y modelado de amenazas
- Autenticación
- Gestión de sesiones

[29] Ibíd., p. 10-11.
[30] Ibíd., p. 22-61.

- Control de acceso
- Manejo de entrada de datos maliciosos
- Criptografía en el almacenamiento
- Gestión y registro de errores
- Protección de datos
- Comunicaciones
- Configuración de seguridad HTTP
- Controles maliciosos
- Lógica del negocio
- Archivos y recursos
- Móvil
- Servicios Web
- Configuración

Al cumplir con estos requisitos en cada fase del desarrollo de software se obtendrá un aplicativo que cumpla con este estándar y, por ende, se considere beneficioso y seguro para la industria y para los clientes.

Dado que cualquier empresa puede implementar la metodología, la misma OWASP la ha integrado a varios de sus proyectos. En concreto, se utiliza en[31]:

- Security Knowledge Framework (SKF). Es una aplicación gratuita creada para entrenar a los desarrolladores en la escritura de código seguro desde el diseño.
- Zed Attack Proxy (ZAP). Consiste en una herramienta para la detección de vulnerabilidades en aplicaciones web, que puede ser utilizada tanto por expertos en seguridad como por desarrolladores nuevos en pruebas de penetración.

[31] Ibíd., p. 21.

- Cornucopia. Es un juego de cartas que ayuda a los desarrolladores de software a identificar requisitos de seguridad a través de metodologías ágiles y tradicionales.

Adicionalmente, la OWASP da unos lineamientos o recomendaciones para cada nivel de verificación, dependiendo de la industria a la que pertenece el cliente:

Figura 16. Recomendaciones de implementación para distintas industrias

Industria	Perfil de amenaza	L1 Recomendación	L2 Recomendación	L3 Recomendación
Salud	La mayoría de los atacantes está buscando información sensible que puede ser utilizada directa o indirectamente para beneficiarse al incluir datos personales a la información de pago. A menudo los datos pueden utilizarse para una variedad de esquemas de fraude, robo de identidad y pagos fraudulentos. Para Los Estados Unidos existen el Health Insurance Portability and Accountability Act (HIPAA) http://www.hhs.gov/ocr/privacy/	Todas las aplicaciones accesibles desde la red	Aplicaciones con cantidades pequeñas o moderadas de información médica confidencial (información de salud protegida), información de identificación personal o datos de pago.	Aplicaciones utilizadas para el control de equipos médicos, dispositivos o registros que pueden poner en peligro la vida humana. Sistemas de pago de Punto de venta y (POS) que contienen grandes cantidades de datos de transacciones que podrían ser utilizados para cometer fraudes. Esto incluye las interfaces administrativas de estas aplicaciones
Venta por menor, alimento, hospitalidad	Muchos de los atacantes en este segmento utilizan tácticas oportunistas de "aplaste y agarre". Sin embargo, también existe una amenaza regular de ataques específicos en aplicaciones que contienen información de pagos, que realizan transacciones financieras o almacenan información personal que pueda ser identificable. Aunque menos probable que las amenazas antes mencionadas, también existe la posibilidad de amenazas más avanzadas las cuales atacan a este segmento de la industria para robar propiedad intelectual, obtener inteligencia competitiva o ganar una ventaja con la organización la cual se ha atacado o de un socio en negociaciones.	Todas las aplicaciones accesibles desde la red.	Adecuado para aplicaciones de negocios, Catálogo de la información del producto, información corporativa interna y aplicaciones con información limitada del usuario (p. ej. información de contacto). Aplicaciones con cantidades pequeñas o moderadas de la funcionalidad de datos o de comprobación de pago.	Sistemas de pago de Punto de venta y (POS) que contienen grandes cantidades de datos de transacciones que podrían ser utilizados para cometer fraudes. Esto incluye las interfaces administrativas de estas aplicaciones. Aplicaciones con un gran volumen de información sensible como números de tarjeta de crédito, nombres completos, documentos de indentidad, etc.

Industria	Perfil de amenaza	L1 Recomendación	L2 Recomendación	L3 Recomendación
Manufacture ra, profesional, transporte, tecnología, utilidades, infraestructu ra y defensa	Estas industrias no parecen tener mucho en común, pero los agentes de amenaza que suelen atacar a las organizaciones en este segmento son más propensos a realizar ataques enfocados con más tiempo, habilidad y recursos. A menudo la información sensible o los sistemas no son fáciles de localizar y requieren utilizar o manipular individuos que trabajen dentro de la organización, utilizando técnicas de ingeniería social. Los ataques pueden involucrar individuos que trabajan dentro de la organización, extraños a la organización, o una combinación de ambos. Sus objetivos pueden incluir acceso a la propiedad intelectual para obtener ventajas estratégicas o tecnológicas. Tampoco queremos pasar por alto a los atacantes que buscan abusar la funcionalidad de la aplicación para influenciar el comportamiento de la aplicación o alterar sistemas sensibles. La mayoría de los atacantes buscan información sensible que puede ser utilizada directa o indirectamente para beneficiarse al incluir datos personales a la información de pago. A menudo los datos pueden utilizarse para una variedad de esquemas de fraude, robo de identidad o pagos fraudulentos.	Todas las aplicaciones accesibles desde la red.	Aplicaciones que contienen información interna o información sobre empleados que pueden aprovecharse utilizando la ingeniería social. Aplicaciones que contiene información poco esencial, pero de importante propiedad intelectual o secretos comerciales.	Aplicaciones que contiene valiosa propiedad intelectual, secretos comerciales o secretos del gobierno (p. ej. en los Estados Unidos esto puede ser cualquier cosa clasificados en secreto o superior) que es fundamental para la supervivencia o el éxito de la organización. Aplicaciones que controlan funcionalidad sensible (p. ej. transporte, fabricación de equipos, sistemas de control) o que tienen la posibilidad de amenazar la seguridad
Financiera y Seguros	Aunque este segmento experimentará intentos de atacantes oportunistas, a menudo es visto como un objetivo de alto valor por atacantes motivados y los ataques se deben muy a menudo a motivos financieros. Comúnmente, los atacantes buscan datos o credenciales de la cuenta que pueden utilizar para cometer fraudes o beneficiarse directamente aprovechando la funcionalidad de flujo de dinero en aplicaciones. Las técnicas incluyen a menudo credenciales robadas, ataques a nivel de aplicación y la ingeniería social. Algunas consideraciones importantes de cumplimiento incluyen EL ESTÁNDAR PCI DSS (PCI DSS), Gramm Leech Bliley Act y Sarbanes-Oxley (SOX).	Todas las aplicaciones accesibles desde la red.	Aplicaciones que contienen información sensible como números de tarjeta de crédito, información personal, que puede mover una cantidad limitada de dinero de manera limitada. Los ejemplos incluyen: (i) transferir dinero entre cuentas en la misma institución o (ii) una forma más lenta del movimiento de dinero (por ejemplo, ACH) con límites de transacción o (iii) transferencias en línea con límites de transferencia dentro de un período de tiempo.	Aplicaciones que contengan grandes cantidades de información sensible o que permiten que sea rápido la transferencia de grandes sumas de dinero (p. ej. transferencias) o transferencia de grandes sumas de dinero en forma de transacciones individuales o como un lote de transferencias pequeñas.

Fuente: The OWASP Foundation, Estándar de Verificación de Seguridad en Aplicaciones 3.0.1.

5.2 CICLO DE VIDA DE DESARROLLO SEGURO DE SOFTWARE (MICROSOFT)

Es una metodología de desarrollo de software impulsada por Microsoft, que permite crear software más seguro reduciendo los costos de desarrollo. Es una de las metodologías más utilizadas debido a su facilidad de uso y al soporte que brinda Microsoft para los desarrolladores que la implementen. Como se observa en la siguiente imagen, tiene una cuota de participación del 15%, ocupando el segundo lugar dentro de las metodologías más utilizadas.

Figura 17. Metodologías de seguridad más utilizadas

Software security	% of respondents
We have developed our own software security methodology	46%
CMM or CMMI	20%
SDL	15%
Other	2%
OpenSAMM	1%
DISA STIG (for operations)	1%
We do not use any such methodology	14%
Don't know	16%

Fuente: http://polux.unipiloto.edu.co:8080/00001900.pdf

Esta metodología consta de 17 prácticas agrupadas en 7 fases secuenciales, las cuales cubren no solo el desarrollo en sí sino también el entrenamiento en seguridad y el establecimiento de un plan de respuesta a incidentes[32]:

[32] IGLESIAS PÉREZ, Julio. Ciclo de Vida de Desarrollo Seguro de Software (es-ES). Microsoft TechNet [en línea]. Disponible en: https://social.technet.microsoft.com/wiki/contents/articles/36676.ciclo-de-vida-de-desarrollo-seguro-de-software-es-es.aspx

Fase 1: Entrenamiento

- Entrenamiento básico de seguridad. Es un requisito previo en la implementación del SDLC. Incluye conceptos como diseño seguro, modelado de amenazas, codificación segura, pruebas de seguridad y mejores prácticas.

Fase 2: Requisitos

- Establecer requisitos de seguridad y privacidad. Facilita la identificación de hitos clave y entregables al definir e integrar los requisitos de seguridad y privacidad desde el principio. También minimiza las interrupciones y permite el cumplimiento de cronogramas.

- Crear puertas de calidad / barras de errores. Al definir los mínimos aceptables de seguridad y calidad se pueden comprender los riesgos de seguridad asociados y se pueden detectar y solucionar los errores de seguridad durante todo el ciclo de desarrollo, aplicando los estándares establecidos.

- Realizar evaluaciones de riesgos de seguridad y privacidad. Examinar los requisitos y costos permite orientar el modelado de amenazas y del diseño a los módulos más significativos y clasificar el impacto de la privacidad en cada módulo.

Fase 3: Diseño

- Establecer requisitos de diseño. Al establecer los requisitos de seguridad y privacidad se minimiza el riesgo de interrupciones en el cronograma y se reducen los gastos.

- Análisis / Reducción de superficies de ataque. Análisis exhaustivo de la superficie general de ataque y restricción de accesos aplicando privilegios mínimos y empleando defensas estratificadas en la medida de lo posible.

- Usar la modelización de amenazas. Estructurar las amenazas durante el diseño permite identificar vulnerabilidades, determinar riesgos y establecer mitigaciones de los mismos de una forma más eficaz y menos costosa.

Fase 4: Implementación

- Utilizar herramientas aprobadas. Usar herramientas y comprobaciones de seguridad aprobadas permite automatizar el proceso y aplicar las mejores prácticas a un costo bajo. Mantener la lista de herramientas actualizada permite utilizar las últimas versiones e incluir nuevas funcionalidades y análisis de seguridad.

- Depreciar funciones inseguras. El análisis de todas las APIs y funciones, descartando aquellas inseguras, ayuda a reducir las vulnerabilidades a muy bajo costo. Se deben utilizar herramientas de análisis de código para verificar el desarrollo y, en caso de ser necesario, reemplazarlo por opciones más seguras.

- Realizar análisis estático. Analizar el código antes de compilarlo asegura el uso de políticas de codificación segura y proporciona un método escalable de revisión del mismo.

Fase 5: Verificación

- Realizar análisis dinámico. Comprobación de la funcionalidad del desarrollo en tiempo de ejecución para detectar problemas de uso de memoria, de privilegios de usuario y otros problemas de seguridad.

- Pruebas de fuzzing. Cargar datos malintencionados para inducir fallos en el software ayuda a detectar problemas de seguridad antes de la liberación con poca inversión de recursos.

- Revisión de superficie de ataque. Asegura que los cambios de diseño o implementación han sido tenidos en cuenta y que los nuevos vectores de ataque producto de estos cambios han sido revisados y mitigados debidamente.

Fase 6: Liberación

- Crear un plan de respuesta a incidentes. La creación de este plan permite enfrentar nuevas amenazas que no hayan sido detectadas previamente. Debe incluir la lista de contactos de emergencia de seguridad y los planes de servicio para el código heredado y para el código de terceros licenciado.

- Realizar la revisión final de seguridad. La revisión de las actividades de seguridad realizadas permite asegurar la disponibilidad del software. Esta incluye examinar el modelo de amenazas, los resultados de las pruebas y el desempeño general contra las puertas de calidad y las barras de fallos anteriormente definidas.

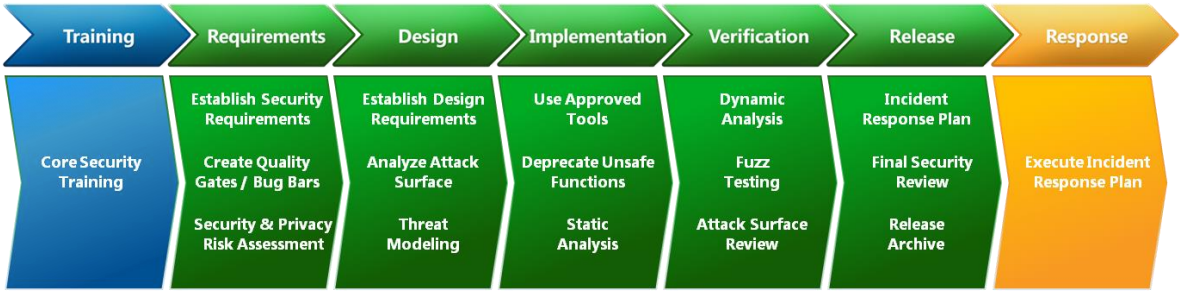
- Certificar lanzamiento y archivado. La certificación garantiza el cumplimiento de los requisitos de seguridad y privacidad exigidos en las fases previas. Archivar la información recolectada durante todo el proceso ayuda en las tareas de mantenimiento posteriores y reduce los costos asociados a la ingeniería de software sostenida.

Fase 7: Respuesta

- Ejecutar un plan de respuesta a incidentes. Implementar el plan definido en la fase anterior ayuda a proteger a los usuarios de las vulnerabilidades de seguridad y/o privacidad que surjan a futuro[33].

En la siguiente imagen se puede observar la secuencialidad de las fases de esta metodología:

Figura 18. Fases del ciclo de vida de desarrollo de software de Microsoft



Fuente: https://www.microsoft.com/en-us/sdl/default.aspx

5.3 CORRECTNESS BY CONSTRUCTION (CbyC)

Es una metodología empleada para el desarrollo de software que necesite alto nivel de seguridad y que éste se pueda demostrar. Pretende una tasa mínima de defectos y gran resiliencia, lo cual se logra con dos premisas: la dificultad para introducir errores y la capacidad de eliminarlos tan pronto sean agregados.

CbyC produce un software correcto desde el principio, con máximos requisitos de seguridad, con detalle en la definición del comportamiento del proceso y con un diseño sólido y verificable.

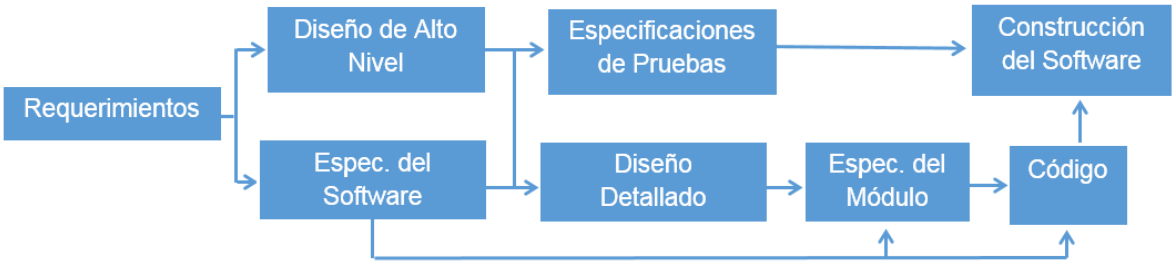
También utiliza conceptos de las metodologías ágiles de desarrollo de software para entregas parciales que puedan ser valoradas y corregidas en poco tiempo[34].

Consta de ocho fases no secuenciales, las cuales se pueden observar en la siguiente imagen:

[33] Ibíd.

[34] BRITO ABUNDIS, Carlos Joaquín. Metodologías para desarrollar software seguro. ReCIBE [en línea]. Disponible en: http://recibe.cucei.udg.mx/revista/es/vol2-no3/computacion05.html

Figura 19. Fases de desarrollo de CbyC



Fuente: http://recibe.cucei.udg.mx/revista/es/vol2-no3/computacion05.html

- Fase de requerimientos. Es donde se especifican los objetivos, las funcionalidades y los requisitos no funcionales del software. Se utilizan diagramas contextuales, de clase y definiciones operativas. Existen procesos de trazabilidad de los requerimientos y de las amenazas correspondientes a cada uno, incluyendo así los requerimientos de seguridad asociados.

- Fase de diseño de alto nivel. Se detalla el proyecto (funcionalidades, estructura de la base de datos, mecanismos de transacciones y comunicaciones) y la interacción entre sus diferentes módulos, y se enfatiza en los procesos de seguridad. Utiliza metodología formal para la definición de alto nivel, obteniendo una descripción, comportamiento y modelo precisos.

- Fase de especificación del software. Documentación formal de las especificaciones de las interfaces de usuario y de los niveles superiores. Se debe realizar un prototipo de desarrollo que incluya estas características, de forma que pueda ser validado por los interesados.

- Fase de diseño detallado. Se relacionan la totalidad de los módulos y procesos del software y su funcionalidad dentro del mismo. Hace uso de la notación formal para evitar ambigüedad en la interpretación de los datos.

- Fase de especificación de los módulos. Definición de estados y comportamientos de cada módulo de acuerdo con el flujo de información. Se debe enfocar cada módulo en el bajo acoplamiento y la alta cohesión, minimizando los problemas cuando se produzca una falla en el flujo de información.

- Fase de codificación. Se deben evitar las ambigüedades en el código, el cual debe ir perfectamente documentado para poder realizar los análisis estáticos necesarios y las revisiones al mismo.

- Fase de las especificaciones de las pruebas. Las pruebas son tan importantes como el desarrollo en sí mismo, por eso deberán estar perfectamente especificadas,

incluyendo todo lo relacionado en las fases anteriores (requerimientos, diseño de alto nivel y especificaciones del software). Se realizan pruebas de valores límite, de comportamiento del software y de requerimientos no funcionales, todas ellas orientadas a las especificaciones y realizadas a nivel de sistema.

- Fase de construcción del software. Se utilizan metodologías ágiles. Desde el comienzo se tiene la estructura completa del software con sus interfaces y mecanismos de comunicación pero con funcionalidad limitada, la cual se va aumentando a medida que se va avanzando en el proyecto[35].

5.4 MODELO DE MADUREZ PARA EL ASEGURAMIENTO DE SOFTWARE (SAMM)

Desarrollado por el OWASP, consiste en un *framework* abierto que permite a las empresas formular y desarrollar una estrategia de seguridad en el software.

Su implementación flexible permite que sea utilizado por empresas de todos los tamaños y para cualquier estilo de desarrollo.

Involucra tres niveles de madurez para cada una de las doce prácticas de seguridad definidas en el modelo, las cuales se agrupan en cuatro funciones de negocio relacionadas con el desarrollo de software. También incluye acciones para medir el desempeño del modelo, entender sus beneficios y estimar los costos asociados[36].

En la siguiente imagen se puede observar la estructura completa de SAMM, con cada una de las funciones de negocio y sus respectivas prácticas de seguridad.

Figura 20. Estructura de SAMM

Fuente: THE OWASP FOUNDATION. Software Assurance Maturity Model Versión 1.0.

[35] Ibíd.

[36] THE OWASP FOUNDATION. Software Assurance Maturity Model Versión 1.0. p. 3.

A continuación se describen las cuatro funciones de negocio que lo componen y las prácticas de seguridad asociadas a cada una:

Gobierno.
Se enfoca en los procesos y actividades de la organización que gestionan las actividades globales de desarrollo de software, como los procesos de negocio establecidos por la organización relacionados con el desarrollo[37].

- Estrategia y métricas. Abarca la dirección estratégica de la empresa en el aseguramiento del software, e incluye los procesos y actividades referentes a la posición de la organización respecto a la seguridad de la información.
- Política y cumplimiento. Establecimiento de las estructuras de control y auditoría, de forma que se cumplan las regulaciones legales en cuanto a aseguramiento, tanto de los desarrollos en proceso como del software terminado.

- Educación y orientación. Capacitar y orientar al personal de desarrollo de software en temas de seguridad relacionados con sus funciones y trabajo[38].

Construcción.
Procesos y actividades orientados a la definición de metas dentro de los proyectos de desarrollo. Incluye la gestión del producto, definición de requisitos de seguridad, arquitectura de alto nivel, diseño detallado e implementación[39].

- Evaluación de amenazas. Identificar y caracterizar los riesgos en el desarrollo del software para adelantar su gestión.

- Requisitos de seguridad. Incluir las necesidades de seguridad en el proceso de desarrollo del software para determinar correctamente su funcionalidad.

- Arquitectura de seguridad. Fortalecer el proceso de diseño incluyendo la seguridad desde el comienzo y en los marcos de trabajo de desarrollo[40].

Verificación.
Se orienta a la forma en que la empresa verifica y prueba sus desarrollos de software. Involucra el aseguramiento de la calidad a través de las pruebas y otras actividades de evaluación[41].

---

[37] Ibíd., p. 8.
[38] Ibíd., p. 9.
[39] Ibíd., p. 8.
[40] Ibíd., p. 9.
[41] Ibíd., p. 8.

- Revisión de diseño. Incluye la validación del software en cuanto a seguridad, de acuerdo con los estándares definidos por la organización.

- Revisión de código. Instrumentos de evaluación del código para la detección de vulnerabilidades y la mitigación de riesgos conforme a las expectativas de la organización.

- Pruebas de seguridad. Realización de pruebas dinámicas en ambientes controlados para liberar versiones con estándares de calidad mínimos establecidos por la empresa[42].

Implementación.
Incluye los procesos y actividades relacionados con la puesta en producción de los productos desarrollados. Incluye la entrega del software a los usuarios finales, la instalación en ambientes internos y externos y la configuración y operaciones en los ambientes de producción[43].

- Administración de vulnerabilidades. Establece procesos y actividades para administrar los reportes sobre las vulnerabilidades detectadas y las acciones tomadas.

- Fortalecimiento de ambientes. Implementa controles para los diferentes ambientes de desarrollo a fin de reforzar la seguridad de las aplicaciones cuando ya se han implementado.

- Habilitación operativa. Recopilación de la información referente a la seguridad en la configuración, instalación y ejecución de los programas de la organización[44].

Para cada una de estas prácticas de seguridad se establecen tres niveles de madurez:

- Nivel 1. Entendimiento básico y provisión inicial de la práctica de seguridad.

- Nivel 2. Aumento de la eficiencia y efectividad de la práctica de seguridad.

- Nivel 3. Completo dominio de la práctica de seguridad[45].

[42] Ibíd., p. 9.
[43] Ibíd., p. 8.
[44] Ibíd., p. 9.
[45] Ibíd., p. 9.

5.5 BUILDING SECURITY IN MATURITY MODEL (BSIMM)

Se basa en un estudio de seguridad de software en empresas del mundo real, con el que se lleva a cabo la medición de las iniciativas y actividades que ha realizado cada una en esta materia. Desde el 2008 hasta ahora se han estudiado 72 iniciativas en pro de la seguridad de cada empresa.

Desde este punto de vista, el objetivo principal de Building Security In Maturity Model es cuantificar las actividades realizadas por las iniciativas adelantadas en empresas reales en cuanto a seguridad del software se refiere[46].

En BSIMM siempre se tiene en cuenta que no todas las empresas necesitan el mismo grado de seguridad para lo que realizan diferentes pruebas y así poder brindar la mejor clasificación de necesidades.

Cualquier persona o entidad que haya creado y ejecutado una iniciativa de seguridad del software puede hacer uso de BSIMM. Normalmente es un ejecutivo de la compañía quien desarrolla este tipo de iniciativas[47].

La metodología utilizada para el desarrollo de BSIMM consta de:

- Conocimiento de las prácticas de seguridad del software para crear modelos de referencia.
- Entrevistas con los ejecutivos a cargo de las iniciativas en cada una de las empresas.
- Creación de fichas de puntaje para cada iniciativa.

Los objetivos que se persiguen con la creación de BSIMM son[48]:

- Proporcionar recursos a las empresas que deseen crear o mejorar la seguridad de software.
- Mejorar la calidad de los códigos de seguridad.
- Aclarar conceptos de las estrategias utilizadas en seguridad de software.
- Toma de decisiones efectivas y a tiempo para la gestión del riesgo.

[46] MCGRAW, Gary. Building Security in Maturity Model Release 5.0. p. 2.
[47] Ibíd., p. 2.
[48] Ibíd., p. 4.

El componente metodológico en que se basa BSIMM se denomina Modelo de Referencia para la Seguridad del Software (MRSS). Este consta de cuatro dominios dentro de los que se desarrollan 12 prácticas (3 por cada dominio)[49]:

- **Gobernanza.** Es el primer dominio y en él son muy importantes tanto la estrategia, la planificación, la asignación de roles, la identificación de metas de seguridad del software y la determinación de presupuestos. En este dominio también es fundamental la capacitación, pues con esta los desarrolladores obtienen el conocimiento integral y la actualización necesaria para poner en práctica en sus compañías. Las tres prácticas asociadas a este dominio son: Estrategia y métricas, Cumplimiento y política, y Capacitación.

- **Inteligencia.** Permite estar muy atentos a cualquier ataque en la seguridad del software, ya que se puede fácilmente pasar como un atacante con la herramienta Modelos de Ataque, ya que captura la información, modela las amenazas, se desarrollan los casos de uso, se realiza la clasificación de datos y se definen los patrones de ataque específicos. También se maneja la práctica Características de Seguridad y Diseño, con la cual se manejan patrones de seguridad más importantes, y la práctica de Normas y Requisitos, donde se plasma toda reglamentación que debe cumplirse en materia normativa y legal.

- **Puntos de Contacto con el SSDL (Secure Software Development Lifecycle).** En este dominio se pueden encontrar dos prácticas de seguridad que son consideradas las más importantes, como son el Análisis de la Arquitectura y la Revisión de Código. Con estas se puede realizar un plan de evaluación y correcciones para la organización, de forma que se realicen las correcciones necesarias y las actualizaciones precisas.

- **Despliegue.** Son pruebas de penetración tercerizadas para obtener un punto de vista externo. Estas pruebas principalmente estudian los puntos más vulnerables de la configuración final y así mismo se retroalimentan directamente con la información obtenida sobre defectos y mitigación de los mismos.

En conclusión, todos y cada uno de los aspectos considerados son indispensables para llevar a cabo el BSIMM y tener éxito total en los modelos de seguridad del software implementados. En todas las compañías debería implementarse un modelo de seguridad del software a la mayor brevedad posible.

[49] Ibíd., p. 21.

La siguiente imagen muestra el esquema general del modelo de referencia sobre el que está basado el BSIMM.

Figura 21. Modelo de Referencia para la Seguridad del Software

Modelo de Referencia para la Seguridad del Software (MRSS)			
Gobernanza	**Inteligencia**	**Puntos de Contacto con el SSDL**	**Despliegue**
Estrategia y Métricas	Modelos de Ataque	Análisis de la Arquitectura	Pruebas de Penetración
Cumplimiento y Política	Características de Seguridad y Diseño	Revisión de Código	Entorno del Software
Capacitación	Normas y Requisitos	Pruebas de Seguridad	Gestión de Configuración y Gestión de Vulnerabilidades

Fuente: McGraw, Gary. Building Security in Maturity Model. Release 5.0

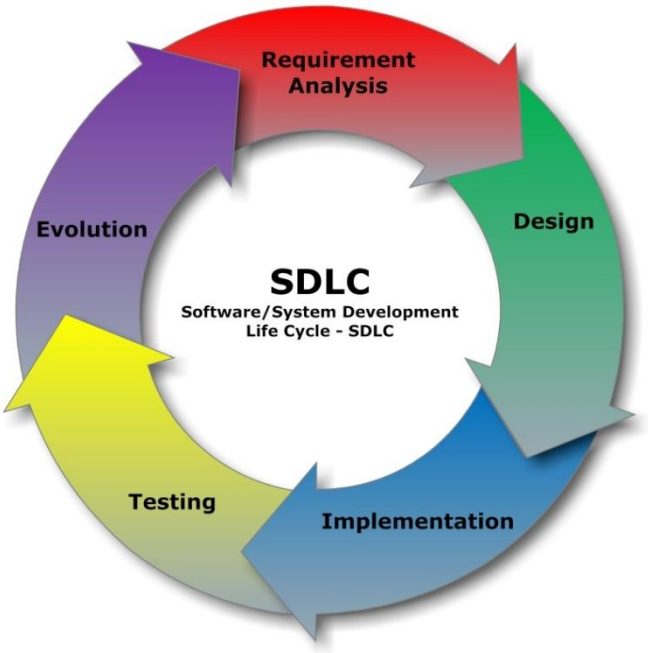
6. ETAPAS DEL CICLO DE VIDA DEL DESARROLLO DE SOFTWARE (SDLC)

El ciclo de vida del desarrollo de software (Software Development Life Cycle – SDLC) es el proceso lógico que determina los pasos que deben seguirse para el desarrollo de un programa informático. Consta de cinco fases básicas, de las cuales las tres primeras suelen demandar la mayor parte del tiempo, pues implican toda la planificación del sistema y es aquí donde se deben realizar la mayoría de cambios y ajustes, pues hacer esta tarea en las etapas posteriores es mucho más complicado y costoso.

En cada una de las etapas es fundamental la interacción con el usuario final, ya que este es finalmente quien determina qué se hace y cómo se debe hacer dentro del software. Este aspecto es clave para el éxito del proyecto. Para esto se pueden utilizar recursos como maquetas de interfaces, diagramas de flujo o mapas del aplicativo, que permitan ver la relación entre los diferentes módulos que conformarán el software[50].

A continuación se puede observar el modelo cíclico del SDLC con sus cinco etapas.

Figura 22. Etapas del SDLC



Fuente: http://wh0s.org/2014/12/07/s-sdlc-aplicando-seguridad-al-ciclo-de-vida-del-software/#respond

[50] FALCONI, David. Ciclo de Vida de Desarrollo de Sistemas – (Systems Development Life Cycle [SDLC]). Tipos de Sistemas de Información [en línea]. Disponible en: https://tiposdesistemasdeinformacion.wordpress.com /2015/04/09/ciclo-de-vida-de-desarrollo-de-sistemas-systems-development-life-cycle-sdlc/

6.1 ANÁLISIS DE REQUERIMIENTOS

Permite establecer las características generales que deben tenerse en cuenta para delimitar el entorno de implementación del software y el alcance del mismo. Entre estas se encuentran[51]:

- Arquitectura de la aplicación: cliente – servidor o aplicativo web.
- Plataforma de instalación: sistema operativo, servidor web, motor de base de datos.
- Plataforma de ejecución: pc, tablet, teléfono celular, diseño responsivo.
- Tipos de datos que se manejan: confidenciales, públicos, cargue de archivos.
- Normas y requerimientos que se deben cumplir.
- Módulos y funcionalidades que se deben desarrollar.
- Flujo de la información dentro del software.
- Registros que se deben generar: impresiones, pdf, gráficos, logs, auditorías.
- Perfiles de usuario que se requiere definir: administrador, editor, consulta.
- Restricciones de acceso a los datos para cada perfil: lectura, escritura, borrado.
- Acciones sobre el sistema: cambiar la configuración, arrancar servicios.
- Modos de autenticación: password, token, biométrico, doble factor.

6.2 DISEÑO

En esta etapa se define la arquitectura del software y las funcionalidades y módulos que lo conforman, así como las interfaces a realizar y los datos a manejar. Es la

[51] MILANO, Pablo. Seguridad en el ciclo de vida del desarrollo de software. Cybsec S.A. [en línea]. Disponible en: http://www.cybsec.com/upload/cybsec_Tendencias2007_Seguridad_SDLC.pdf, p. 5, 6.

implementación de los requerimientos definidos en la etapa anterior. Entre las consideraciones a tener en cuenta en esta etapa, se encuentran[52]:

- Diseño de los mecanismos de autenticación.
- Diseño de los mensajes de error.
- Interacción con las plataformas y dispositivos de seguridad.
- Administración de la información sensible.
- Diseño de la auditoría y logs.
- Separación de privilegios de los usuarios.
- Análisis del riesgo.

6.3 IMPLEMENTACIÓN

En esta etapa es donde se lleva a cabo la creación del código fuente del programa. Acá se plasma todo lo definido en las etapas anteriores, y es donde se puede empezar a mostrar al cliente cómo va quedando el desarrollo. Se pueden realizar prototipos o sistemas de prueba que permitan comprobar y depurar el software a medida que se construye. En esta etapa de debe tener en cuenta[53]:

- Validación de los datos de entrada.
- Validación de los datos en todo el software.
- Definición de los tipos de datos a utilizar y su tamaño.
- Definición del tipo de codificación de los caracteres utilizados.
- Eliminar o restringir los caracteres especiales.
- Utilizar procedimientos almacenados en lugar de sentencias SQL dinámicas.

[52] Ibíd., p. 8.
[53] Ibíd., p. 14.

- Separar los datos del código.
- Manejo de errores y logs.
- Manejo de archivos.
- Manejo de memoria.
- Utilización de funciones y APIs comprobadas[54].

6.4 PRUEBAS

Es la etapa donde se prueba todo el código realizado, tanto en su estructura (pruebas estáticas) como en su funcionamiento (pruebas dinámicas). Para este proceso existen múltiples herramientas que se pueden utilizar para realizar diferentes tipos de pruebas y que permiten detectar diferentes tipos de vulnerabilidades. Así mismo, es recomendable que el usuario final también realice pruebas sobre el software para que ayude en la estructuración de las interfaces y verifique el correcto flujo y manejo de los datos dentro del mismo.

Por último, es altamente recomendable que las pruebas sean realizadas por personal diferente al desarrollador, o incluso por una entidad externa, lo que permite independencia y mayor objetividad en este proceso y ayuda a detectar fallas que el desarrollador, al estar inmerso en el código, no podría detectar.

Algunas de las cosas que se pueden probar, son:

- Requerimientos de autenticación.
- Requerimientos de complejidad de contraseñas.
- Requerimientos para el bloqueo de cuentas.
- Verificación de restricciones de acceso.
- Mecanismos de registro y logueo.

[54] CUENCA DÍAZ, César. Desarrollo Seguro: Principios y Buenas Prácticas. OWASP [en línea]. Disponible en: https://www.owasp.org/images/9/93/Desarrollo_Seguro_Principios_y_Buenas_Pr%C3%A1cticas..pdf

- Mensajes de error personalizados[55].

6.5 EVOLUCIÓN Y MANTENIMIENTO

En esta última etapa se verifican todos los procesos anteriores y se corrigen las fallas que no se hayan detectado antes. Debido a que el proceso es cíclico, se pueden volver a desarrollar todas las etapas en el evento de agregar una nueva funcionalidad al software o cuando se requiera un desarrollo que reemplace a uno anterior. Es importante que se haga este seguimiento aun cuando se tenga el producto terminado y se utilice constantemente, pues diariamente aparecen nuevas amenazas o se descubren nuevas vulnerabilidades que afectan el software. Entre las actividades a tener en cuenta, se encuentran:

- Eliminación de servicios innecesarios.
- Reconfiguración y actualización del entorno.
- Manejo de versiones.
- Actualización de parches y releases[56].
- Licenciamiento y derechos de autor.
- Mantenimiento de bases de datos.
- Mantenimiento de certificados digitales.
- Realización periódica de backups.
- Migraciones e interdependencias.
- Finalización de uso del software por obsolescencia.

[55] MILANO. Op. cit., p. 16.
[56] Ibíd., p. 19.

7. TÉCNICAS Y PROCEDIMIENTOS DE SEGURIDAD APLICABLES AL SDLC

Dentro de cada etapa del ciclo de vida de desarrollo de software (SDLC) existen ciertas técnicas y/o procedimientos que se pueden llevar a cabo para lograr la implementación de la seguridad. Algunas de estas técnicas se definen en las metodologías relacionadas en el capítulo 5 del presente documento, por lo que solo se relacionarán de manera general.

7.1 REVISIONES E INSPECCIONES MANUALES

Consiste en realizar un levantamiento de información que permita determinar, desde la etapa de requerimientos, las necesidades de seguridad que cada grupo de interés pueda tener respecto al desarrollo de software, y al mismo tiempo evaluar cómo se ha ejecutado el desarrollo.

Puede basarse en las siguientes actividades[57]:

- **Entrevistas.** Consiste en preguntar a cada uno de los interesados, mediante una reunión o a través de un formulario, cuáles son las necesidades de seguridad que considera deben ser satisfechas dentro del desarrollo del software. Una vez realizadas todas las entrevistas, se hace una tabulación para determinar cuáles son las necesidades más importantes y de esta manera priorizarlas en los requerimientos a tener en cuenta. También se realizan en otras etapas del SDLC para conocer el nivel de conformidad de los interesados con respecto al desarrollo y despliegue del software.

- **Análisis de documentación.** Se debe realizar la documentación generada en la planificación, para determinar qué requerimientos de seguridad se tuvieron en cuenta durante el levantamiento de esta información. Con esto se tiene una primera aproximación a lo que el solicitante desea en esta materia, o si definitivamente no lo ha tenido en cuenta en sus requerimientos iniciales. En etapas posteriores, permite validar la ejecución del desarrollo en términos de la seguridad implementada.

- **Evaluación del SDLC.** En etapas posteriores del SDLC, las revisiones e inspecciones manuales permiten determinar el nivel de implementación de la seguridad dentro del ciclo y cómo se ha visto afectado el proyecto en cuanto a su ejecución, en la medida que se hayan detectado y corregido las fallas.

Dentro de las ventajas que presenta esta técnica, se encuentran:

[57] AGUILERA DÍAZ, Vicente. Técnicas de Evaluación de Seguridad en el Software. ITSMF, 2017. p. 16

- No requiere soporte de tecnología. Al ser una actividad netamente manual, no es necesaria ninguna herramienta tecnológica especializada para llevar a cabo esta técnica.

- Flexibilidad. Permite realizar tantos análisis, cuestionarios y evaluaciones como se requieran para asegurar la correcta implementación de la seguridad en el software, y estos pueden ser tan flexibles y variados como el proyecto lo permita.

- Trabajo en equipo. Al involucrar a todos los interesados, se conocen los diferentes puntos de vista respecto al proyecto y se pueden realizar mesas de trabajo que permitan llevar a consensos dentro de las diferentes áreas y personal.

- Se utiliza desde las fases tempranas del SDLC. Coadyuva en el levantamiento de requerimientos y se puede utilizar también en el diseño y en las pruebas.

Pero presenta las siguientes desventajas[58]:

- Puede consumir mucho tiempo. Dependiendo del número de interesados en el proyecto, las entrevistas y los cuestionarios pueden ser dispendiosos ya que dependen del tiempo disponible de cada persona. Así mismo, la cantidad de documentación que se requiera analizar puede ser enorme y, por ende, complicada de manipular.

- Indisponibilidad del recurso. Muchas veces no se cuenta con el tiempo de la persona o la documentación no está a la mano para poder realizar el proceso de forma ágil.

- Requiere habilidades específicas. Las habilidades blandas (soft skills) son importantes para esta técnica pues se requiere poder de comunicación y empatía con los interesados para lograr resultados efectivos, tanto en las entrevistas como en la obtención de datos y documentos necesarios para el proyecto.

7.2 MODELADO DE AMENAZAS

Consiste en estructurar la información (capturar, organizar y analizar) para determinar en qué medida puede afectar la seguridad del desarrollo, y de esta manera tomar decisiones basadas en ese análisis y en la identificación de los riesgos de seguridad[59].

Algunas de las actividades que pueden incluirse, son:

[58] Ibíd., p. 18.
[59] WHOS. S-SDLC: Aplicando seguridad al ciclo de vida del software. WHOS [en línea]. Disponible en: https://wh0s.org/2014/12/07/s-sdlc-aplicando-seguridad-al-ciclo-de-vida-del-software/

- **Descomponer la aplicación.** Al determinar cada una de las funcionalidades y flujo de datos que componen el desarrollo, será más fácil determinar los riesgos de seguridad en cada uno de ellos.

- **Clasificación de los activos de información.** Identificar los activos de información permite definir prioridades a la hora de establecer los esquemas de seguridad.

- **Identificación de las vulnerabilidades y amenazas potenciales.** Al determinar los problemas conocidos en el entorno de desarrollo (servidores, sistemas operativos, motores de bases de datos, lenguajes de programación, etc.) se pueden atacar de forma temprana y así minimizar sus efectos al realizar un aseguramiento a la medida.

- **Definir las estrategias de mitigación.** Se deben definir los controles necesarios para mitigar las vulnerabilidades y amenazas detectadas[60].

Ventajas[61]:

- Punto de vista del atacante. Se puede realizar un análisis desde la visión del atacante, actuando de la forma en que él lo haría.

- Toma de decisiones informadas. Al conocer las vulnerabilidades y las amenazas, se puede orientar las decisiones respecto a la seguridad del proyecto.

- Flexibilidad. Se puede implementar la seguridad tan profundo como se requiera y de acuerdo a los análisis efectuados.

- Implementación desde las primeras fases del ciclo. Se pueden definir y modelar las amenazas desde la etapa de requerimientos del SDLC.

Desventajas[62]:

- Conocimiento profundo del desarrollo. Se requiere conocer todo el entorno de implementación y desarrollo del software para determinar las amenazas que lo puedan afectar.

- Proceso continuo. La revisión y modelado de amenazas debe hacerse en todas las etapas del SDLC, y requiere la actualización del proceso en la medida que se vayan mitigando los riesgos.

[60] AGUILERA. Op. Cit., p. 19.
[61] Ibíd., p. 20.
[62] Ibíd., p. 21.

7.3 REVISIÓN DEL CÓDIGO DESARROLLADO

Se requiere hacer pruebas estáticas para determinar la robustez del código y de esta manera validarlo dentro del esquema de seguridad del proyecto. También se requiere conocer a fondo la funcionalidad del aplicativo para saber cómo buscar y detectar las posibles vulnerabilidades[63].

Esta técnica es muy completa y efectiva, en la medida que existen herramientas que pueden automatizar el proceso y detectar la mayor parte de amenazas. También es muy precisa, pues permite determinar cada amenaza conforme al enforno en que se encuentra montado el software y a sus características específicas[64].

Sin embargo, presenta inconvenientes como un conocimiento profundo en desarrollo de software y en el manejo de las herramientas que se vayan a utilizar, para sacar el mejor provecho de ellas. Así mismo, no permite detectar los errores en tiempo de ejecución, y se debe manejar correctamente el versionamiento para que el código analizado sea igual al código desplegado[65].

7.4 PRUEBAS DE INTRUSIÓN

Permiten realizar las pruebas dinámicas del software, de forma que se vea su funcionamiento en el entorno en que va a trabajar. De esta forma se visualiza cómo un atacante podría explotar las vulnerabilidades del software y así mitigar estos ataques.

Entre las ventajas de esta técnica, se encuentra que puede ser muy rápida en la detección de errores ya que no se requiere ser un experto informático para realizar las pruebas necesarias, y además existen las herramientas que permiten automatizar el proceso. El código que se evalúa es el que puede ver un atacante y por tanto no se requiere la revisión del código completo.

Como desventajas, se encuentra que no se puede aplicar en las etapas tempranas del SDLC ya que se requiere que el software ya se encuentre desarrollado, así sea en parte, y además solo evalúa la funcionalidad accesible, por lo que no es una prueba del código completo sino únicamente lo que se puede visualizar a través del cliente[66].

---

[63] Ibíd., p. 22.
[64] Ibíd., p. 23.
[65] Ibíd., p. 24.
[66] Ibíd., p. 27.

8. RESULTADOS OBTENIDOS

De acuerdo con las diferentes metodologías encontradas para la implementación de la seguridad dentro del ciclo de vida de desarrollo del software, se observa que cada una tiene sus particularidades y procedimientos, de forma que se pueden ajustar a diferentes esquemas y estilos de desarrollo. De esta forma se puede determinar la aplicabilidad de cada metodología y su adopción por parte de las empresas de desarrollo de software en Colombia.

El Estándar de Verificación de Seguridad en Aplicaciones - ASVS, al ser un compilado general de requerimientos y pruebas que pueden realizarse, y por su facilidad de uso, tiene un amplio abanico de posibilidades para ser utilizado no solo por los desarrolladores de software, sino también por arquitectos de software, *testers*, profesionales de seguridad de la información e, incluso, por los clientes, quienes pueden probar de esta forma la seguridad de la aplicación que han adquirido.

Esa versatilidad también permite que el estándar sea aplicable a diferentes tipos de aplicaciones, no solo aplicaciones web, sino aplicaciones cliente, *web services*, aplicaciones móviles o cualquier tipo de aplicación moderna.

Los niveles de seguridad definidos en ASVS también permiten tipificar el alcance de seguridad necesario para determinado tipo de aplicación, de forma que no se requiera definir e implementar todos los controles para todas las aplicaciones, sino que el desarrollador tiene la posibilidad de seleccionar el nivel de seguridad deseado y, aun así, cumplir con el estándar.

En cuanto al Ciclo de Vida de Desarrollo Seguro de Software de Microsoft, este puede ser utilizado por empresas y desarrolladores de software que ya tengan implementado un ciclo de vida de desarrollo de software, pues presenta requisitos y pruebas de cumplimiento de seguridad para cada una de las etapas del ciclo y además abarca otras fases adicionales que complementan el proceso, tales como el entrenamiento básico previo al desarrollo, el mantenimiento del software y la elaboración de un plan de respuesta a incidentes.

Su objetivo es implementar la seguridad al tiempo que se reducen los costos del desarrollo, por esto abarca todo el ciclo desde su primera fase, ya que está comprobado que esta es la forma más económica de implementar la seguridad. Por esto es ideal para empresas que ya tengan una metodología estructurada y donde el componente de costos sea un factor preponderante en la comercialización del producto final.

Correctness by Construction (CbyC), por su parte, puede ser implementado por casas de software que requieran altos índices de seguridad, como por ejemplo para el desarrollo de aplicaciones de sectores críticos como banca, aeronáutica, medicina e infraestructuras nucleares. Esta rigurosidad en los requerimientos hace que su uso no sea recomendado para cualquier desarrollador, y ciertamente requiere profesionales especializados en desarrollos para el campo de aplicación en el cual se va a utilizar el software, y un músculo financiero importante para trabajar a este nivel de especialización y poder cumplir a grandes clientes de estos sectores.

Del Modelo de Madurez para el Aseguramiento del Software (SAMM), se puede decir que se puede ajustar a las necesidades específicas de cada empresa ya que se enfoca en mejorar los niveles de seguridad ya existentes, por lo que se puede considerar un complemento a lo ya definido en los otros modelos. Funciona en empresas de cualquier tamaño siendo aplicable a toda la organización o solo a una parte de ella.

Al determinar el nivel de madurez de la empresa en cuanto a la seguridad del software, permite definir metas a largo plazo que conllevan al mejoramiento de la seguridad a medida que se van implementando los controles, hasta llegar al nivel de madurez deseado por la empresa.

Una de sus premisas indica que no hay una sola receta que funcione para todas las organizaciones. Esta flexibilidad permite su aplicación en cualquier tipo de empresa que desarrolle software, siendo ideal para procesos de certificación ya que induce a la mejora continua.

El BSIMM (Building Security in Maturity Model) es una metodología similar a la anterior, en el sentido de que define el nivel de madurez de una empresa en cuanto a la seguridad informática. La principal diferencia es que BSIMM determina este nivel de madurez al comparar la empresa con los datos obtenidos de otras empresas que implementan la seguridad en sus procesos de desarrollo, permitiendo identificar las metas a las que puede llegar la empresa y las tareas que tiene que cumplir para alcanzarlas.

De esta forma, puede ser aplicada por cualquier empresa de desarrollo que quiera saber en qué nivel de madurez se encuentra con respecto a otras empresas del sector.

Como se puede observar, todas las metodologías estudiadas (a excepción de CbyC) son complementarias entre sí y pueden ser usadas por cualquier tipo de empresa de desarrollo de software que quiera llevar a cabo un proceso serio y concienzudo de implementación de la seguridad. Ya no hay excusas que impidan la implementación de esquemas de seguridad durante el ciclo de vida de desarrollo de software, y la empresa podrá ofrecer productos de calidad y seguros a un costo mínimo en comparación con la gran cantidad de beneficios que puede llegar a obtener.

A continuación se presenta una tabla resumen de las cinco metodologías analizadas, de forma que se puedan apreciar sus principales características.

Tabla 1. Características de las metodologías de aseguramiento de software

Metodología	Características
Estándar de Verificación de Seguridad en Aplicaciones - ASVS	• Es un marco de referencia desarrollado por la OWASP para determinar los requisitos y controles de seguridad necesarios al diseñar, desarrollar, probar y mantener aplicaciones web. • Debido a su facilidad de uso, puede ser utilizado por desarrolladores de software, arquitectos de software, *testers*, profesionales de seguridad de la información e, incluso, por los clientes. • Cuenta con tres niveles de verificación, dependiendo de las necesidades de seguridad requeridas: oportunista, estándar y avanzado. • Su versatilidad permite que pueda usarse no solo en aplicaciones web, sino también en aplicaciones cliente, *web services*, aplicaciones móviles o cualquier tipo de aplicación moderna. • Permite definir el nivel de seguridad deseado para la aplicación, de forma que el desarrollador no requiere implementar todos los controles para cumplir con el estándar. • Para el nivel avanzado se debe cumplir con 181 requisitos de verificación, mientras que para el nivel oportunista solo se requiere cumplir con 87. • Los 181 requisitos de verificación se encuentran agrupados dentro de 16 módulos diferentes.
Ciclo de Vida de Desarrollo Seguro de Software de Microsoft - MSDLC	• Aplica para empresas y desarrolladores de software que ya tengan implementado un ciclo de vida de desarrollo de software en su trabajo. • Consta de 17 prácticas agrupadas en 7 fases, dentro de las cuales se consideran las 5 fases del ciclo de vida de desarrollo de software. • Abarca todas las etapas del SDLC e incluye otras fases adicionales que complementan el proceso, tales como el entrenamiento básico previo, el

Metodología	Características
	mantenimiento del software y la elaboración de un plan de respuesta a incidentes. • Su objetivo es implementar la seguridad al tiempo que se reducen los costos de desarrollo.
Correctness by Construction - CbyC	• Se utiliza en el desarrollo de aplicaciones que requieren altos índices de seguridad, como por ejemplo en sectores críticos como banca, aeronáutica, medicina e infraestructuras críticas. • Su uso no se recomienda para cualquier desarrollador, pues requiere de profesionales especializados en desarrollos para el campo de aplicación en el cual se va a utilizar el software. • Utiliza conceptos de las metodologías ágiles de desarrollo de software para que las entregas parciales puedan ser corregidas en poco tiempo. • Consta de ocho fases no secuenciales, pues algunas se pueden realizar en paralelo y otras de forma iterativa. • Requiere de un recurso financiero importante para trabajar a este nivel de especialización y así poder cumplir a los grandes clientes de estos sectores.
Modelo de Madurez para el Aseguramiento del Software - SAMM	• Se enfoca en mejorar los niveles de seguridad ya existentes, por lo que puede considerarse un complemento a lo ya definido en los otros modelos. • Involucra cuatro funciones de negocio relacionadas con el desarrollo de software, dentro de las cuales se ubican doce prácticas de seguridad (tres por cada función). • Funciona en empresas de cualquier tamaño, siendo aplicable a toda la organización o solo a una parte de ella. • Involucra tres niveles de madurez en cada práctica de seguridad, de forma que el desarrollo se pueda ubicar en alguno de esos niveles. • Permite definir metas a largo plazo que conllevan al mejoramiento de la seguridad a medida que se

Metodología	Características
	van implementando los controles, hasta llegar al nivel de madurez deseado por la empresa. • Permite su aplicación en cualquier tipo de empresa de desarrollo de software, siendo ideal para procesos de certificación ya que induce a la mejora continua.
Building Security in Maturity Model - BSIMM	• Determina el nivel de madurez de la empresa en desarrollo seguro, al compararla con los datos obtenidos de otras empresas que ya implementan la seguridad en sus procesos de desarrollo. • Su objetivo es cuantificar las actividades adelantadas por empresas reales en las iniciativas referentes a la seguridad del software. • Permite identificar las metas a las que puede llegar la empresa y las tareas que tiene que cumplir para alcanzarlas. • Consta de cuatro dominios dentro de los que se desarrollan 12 prácticas (3 por cada dominio). • Permite la toma de decisiones efectivas y a tiempo para la gestión del riesgo en el desarrollo de software.
Fuente Elaboración propia	

Tabla 2. Comparativa entre las metodologías de aseguramiento de software

	ASVS	MSDLC	CbyC	SAMM	BSIMM
Aplica para cualquier tipo de desarrollo	Sí	Sí	No	Sí	Sí
Orientado al ciclo de desarrollo	Sí	Sí	No	No	No
Número de requisitos a cumplir	181	17	8	12	12
Módulos o dominios involucrados	16	7	8	4	4
Requiere poca inversión de recursos	Sí	Sí	No	Sí	Sí
Enfocado a toda la empresa	No	Sí	No	Sí	Sí
Niveles de madurez ofrecidos	3	1	1	3	3
Fácil de implementar	Sí	Sí	No	Sí	No
Fuente Elaboración propia					

9. CONCLUSIONES

- El desarrollo de software es una actividad que debe implementar la seguridad informática en todas sus etapas. Esto es especialmente importante hoy en día debido al auge en la cantidad y refinamiento de los ataques informáticos.

- Las empresas dedicadas a esta actividad deben ser conscientes de la necesidad de establecer una metodología para dicha implementación, ya que esto redundará en la satisfacción de sus clientes y en mejorar la calidad de sus productos.

- La comparación entre las diferentes metodologías presentadas permite deducir que no existe una única forma de implementar un esquema de seguridad en el ciclo de vida de desarrollo de software, sino que cada empresa, acorde a sus características y objetivos, puede realizar los ajustes que considere convenientes y utilizar la metodología que mejor se acomode a sus procesos.

- Existe una gran preocupación a nivel mundial respecto a la seguridad en el desarrollo de software. Es así como grandes empresas como Microsoft, Oracle o IBM tienen definidos sus propios estándares de implementación de la seguridad, e iniciativas colectivas sin ánimo de lucro como la Fundación OWASP también actualizan su información y mejoran sus prácticas para que todos los desarrolladores puedan utilizar sus herramientas y guías de forma libre y gratuita para el aseguramiento del software.

- A medida que la tecnología va evolucionando, así mismo evolucionan los ataques a los sistemas informáticos. Se ha pasado de ataques con virus y gusanos a ataques de ransomware, minado de criptomonedas y redes de bots controladas remotamente. Por eso es importante conocer las principales amenazas que afectan a las aplicaciones on line y cómo se pueden prevenir. En este sentido, el Top 10 de OWASP es la principal guía para obtener esta información y así orientar los esquemas de seguridad hacia la prevención de este tipo de riesgos.

10. RECOMENDACIONES

- Toda compañía que se dedique al desarrollo de software debe tener en su nómina el personal calificado para establecer los esquemas, las metodologías y los procesos en cuanto a la seguridad de la información.

- Debe implementarse un programa de capacitación continua a los desarrolladores de software sobre temas de seguridad de la información y su impacto e implementación en los procesos de desarrollo.

- Se deben implementar los sistemas de seguridad en el ciclo de vida de desarrollo de software con el fin de evitar riesgos innecesarios y mejorar la calidad de los productos y la satisfacción del cliente.

- Toda persona que se dedique al desarrollo de software debe tener y conocer las herramientas que le permitan hacer pruebas de penetración a sus desarrollos, con el fin de detectar las fallas y corregirlas a tiempo.

- Se debe contar con los entornos necesarios para el aseguramiento de la calidad en el software. Estos entornos deben ser: desarrollo, pruebas y producción, y se deben establecer los protocolos y condiciones para que el código desarrollado pueda pasar de un entorno a otro.

- No importa ni el tamaño de la empresa ni su recurso económico, o si se es un desarrollador freelance que trabaja por cuenta propia. Todas las personas involucradas en los procesos de desarrollo de software deben concientizarse sobre la importancia de implementar la seguridad en el software desde las primeras fases del ciclo de desarrollo.

- Es muy importante que los desarrolladores integren alguna de las metodologías analizadas en este documento al ciclo de desarrollo de software para que sus productos cumplan con los estándares de seguridad necesarios y los clientes tengan la confianza en que están adquiriendo una herramienta de calidad y segura.

BIBLIOGRAFÍA

AGUILERA DÍAZ, Vicente. Técnicas de Evaluación de Seguridad en el Software. Catalunya: ITSMF, 2017. 42 p.

BORAL, Sayak. Nginx vs. Apache: ¿Cuál es el mejor servidor? MasGNULinux [en línea], enero de 2019 [revisado 20 de agosto de 2019]. Disponible en Internet: https://maslinux.es/nginx-vs-apache-cual-es-el-mejor-servidor/

BRITO ABUNDIS, Carlos Joaquín. Metodologías para desarrollar software seguro. ReCIBE, Año 2 No.3 [en línea], diciembre de 2013 [revisado 20 de octubre de 2018]. Disponible en Internet: http://recibe.cucei.udg.mx/revista/es/vol2-no3/computacion05.html

CARDADOR CABELLO, Antonio Luis. Implantación de aplicaciones web en entornos internet, intranet y extranet. Málaga: IC Editorial, 2014. 328 p.

CASTRO, Luis. ¿Qué es SSL o TLS? About Español [en línea], 14 de febrero de 2017 [revisado 20 de septiembre de 2018]. Disponible en Internet: https://www.aboutespanol.com/que-es-ssl-o-tls-157625

CATOIRA, Fernando. Vulnerabilidades, estadísticas e impacto. Welivesecurity [en línea], 31 de julio de 2012 [revisado 20 de agosto de 2019]. Disponible en Internet: https://www.welivesecurity.com/la-es/2012/07/31/vulnerabilidades-estadisticas-e-impacto/

CUENCA DÍAZ, César. Desarrollo Seguro: Principios y Buenas Prácticas. OWASP [en línea], s.f. [revisado 7 de noviembre de 2018]. Disponible en Internet: https://www.owasp.org/images/9/93/Desarrollo_Seguro_Principios_y_Buenas_Pr%C3%A1cticas..pdf

DE LUZ, Sergio. mitmAP: Un programa todo en uno para realizar ataques Man In The Middle. Redes Zone [en línea], 6 de diciembre de 2016 [revisado 20 de septiembre de 2018]. Disponible en Internet: https://www.redeszone.net/2016/12/06/mitmap-programa-uno-realizar-ataques-man-in-the-middle/

DÍAZ CHAPARRO, Luis Roberto. Desarrollo de software seguro. Universidad Piloto de Colombia [en línea], sin fecha [revisado 20 de agosto de 2019]. Disponible en Internet: http://polux.unipiloto.edu.co:8080/00001900.pdf

ENISA. ENISA Threat Landscape Report 2018. ENISA [en línea], 28 de enero de 2019 [revisado 20 de agosto de 2019]. Disponible en Internet: https://www.enisa.europa.eu/publications/enisa-threat-landscape-report-2018

FALCONI, David. Ciclo de Vida de Desarrollo de Sistemas – (Systems Development Life Cycle [SDLC]). Tipos de Sistemas de Información [en línea], 9 de abril de 2015 [revisado 7 de noviembre de 2018]. Disponible en Internet: https://tiposdesistemasdeinformacion. wordpress.com/2015/04/09/ciclo-de-vida-de-desarrollo-de-sistemas-systems-development -life-cycle-sdlc/

GIUSTO BILIC, Denise. 10 consejos para el desarrollo seguro de aplicaciones. Welivesecurity by Eset [en línea], 12 de marzo de 2015 [revisado 8 de septiembre de 2018]. Disponible en Internet: https://www.welivesecurity.com/la-es/2015/03/12/10-consejos-desarrollo-seguro-de-aplicaciones/

IGLESIAS PÉREZ, Julio. Ciclo de Vida de Desarrollo Seguro de Software (es-ES). Microsoft TechNet [en línea], 3 de enero de 2017 [revisado 10 de octubre de 2018]. Disponible en Internet: https://social.technet.microsoft.com/wiki/contents/articles/36676. ciclo-de-vida-de-desarrollo-seguro-de-software-es-es.aspx

INSTITUTO COLOMBIANO DE NORMAS TÉCNICAS Y CERTIFICACIÓN. Documentación. Presentación de trabajos académicos. NTC 1486. Séptima actualización. Bogotá D.C.: El Instituto, 2018. 52 p.

--------. Referencias bibliográficas. Contenido, forma y estructura. NTC 6166. Bogotá D.C.: El Instituto, 2016. 60 p.

MCAFEE LABS. Informe de McAfee Labs sobre amenazas – Marzo de 2018. McAfee Labs [en línea] [revisado 20 de agosto de 2019]. Disponible en Internet: https://www.mcafee.com/enterprise/es-es/assets/reports/rp-quarterly-threats-mar-2018.pdf

MCGRAW, Gary. Building Security in Maturity Model. BSIMM-V. Release 5.0. BSIMM, 2013. 69 p.

MICROSOFT CORPORATION. Finalizó el soporte para Windows XP. Microsoft [en línea] [revisado 20 de septiembre de 2018]. Disponible en Internet: https://www.microsoft.com/es-xl/windowsforbusiness/end-of-xp-support

MILANO, Pablo. Seguridad en el ciclo de vida del desarrollo de software. Cybsec S.A. [en línea], 12 de septiembre de 2007 [revisado 7 de noviembre de 2018]. Disponible en Internet: http://www.cybsec.com/upload/cybsec_Tendencias2007_Seguridad_SDLC.pdf

MONTES DÍAZ, María José. Las principales vulnerabilidades web. Hacking Ético [en línea], 4 de abril de 2017 [revisado 8 de septiembre de 2018]. Disponible en Internet: https://hacking-etico.com/2017/04/04/las-principales-vulnerabilidades-web/

SÁNCHEZ, César. Protegiendo tu sitio web de las 10 vulnerabilidades más comunes. Novacreations [en línea], 13 de febrero de 2013 [revisado 20 de septiembre de 2018]. Disponible en Internet: http://novacreations.net/diez-vulnerabilidades-aplicaciones-web/

PIZARRO, Valeria. Citas y Referencias. Referencias electrónicas en normas ICONTEC. Normas Icontec 2018 [en línea], 8 de marzo de 2017 [revisado 31 de agosto de 2018]. Disponible en Internet: http://www.normasicontec.org/referencias-electronicas-normas-icontec/

The OWASP Foundation. Cross-Site Request Forgery (CSRF). OWASP [en línea], 6 de marzo de 2018 [revisado 24 de septiembre de 2018]. Disponible en Internet: https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)

--------. Estándar de Verificación de Seguridad en Aplicaciones 3.0.1. OWASP, 2017. 75 p.

--------. OWASP Top 10 - 2017. Los diez riesgos más críticos en Aplicaciones Web. OWASP, 2017. 25 p.

--------. Software Assurance Maturity Model. Una guía para integrar seguridad en el desarrollo de software. Versión 1.0. OWASP, 2009. 96 p.

WHITE HAT AND OTHER STUFF. S-SDLC: Aplicando seguridad al ciclo de vida del software. WHOS [en línea], 7 de diciembre de 2014 [revisado 17 de noviembre de 2018]. Disponible en Internet: https://wh0s.org/2014/12/07/s-sdlc-aplicando-seguridad-al-ciclo-de-vida-del-software/