INNOVA Research Journal, ISSN 2477-9024
(Septiembre-Diciembre 2020). Vol. 5, No.3.1 pp. 263-280
DOI: https://doi.org/10.33890/innova.v5.n3.1.2020.1440
URL: http://revistas.uide.edu.ec/index.php/innova/index
Correo: innova@uide.edu.ec

Método para el desarrollo de software seguro basado en la ingeniería de software y ciberseguridad

Method for the development of secure software based on software engineering and cybersecurity

Diana María López Álvarez
https://orcid.org/0000-0003-2457-7683
Universidad ECOTEC, Ecuador

Autor para correspondencia: dlopez@ecotec.edu.ec
Fecha de recepción: 27 de mayo de 2020 - Fecha de aceptación: 30 de septiembre de 2020

Resumen
El incremento de los ataques informáticos va en creciente demanda, hoy en día es más frecuente los *ciberdelitos* a los que nuestros datos están expuestos. Por otro lado, el manejo que se tiene del software y las herramientas utilizadas en informática es lo que representa hoy en día un pilar fundamental en lo que enseñanza e investigación científica se refiere, sin embargo, el concepto va mucho más allá de ello, de la mano de la seguridad informática que involucra muchos más factores de los que se creen, no solo físicos sino lógicos, refiriéndose a todas las aplicaciones que ejecutan en cada uno de los equipos y que forman parte del software. Este trabajo presenta un modelo de software seguro basado en el ciclo de vida de desarrollo de software y pretende crear profesionales capaces de desarrollar productos seguros de software y al mismo tiempo crear conciencia ante los diversos problemas que se pueden originar al no establecer medidas de seguridad en el desarrollo de sistemas. Se realiza un análisis final de la importancia que tiene crear profesionales con conocimiento del uso y aplicación de la seguridad informática.
Palabras claves: Ingeniería de software; seguridad informática; seguridad de sistemas; ciber-delitos; herramientas de seguridad.

Abstract
The increase in computer attacks is in increasing demand, nowadays cybercrime to which our data is exposed is more frequent. On the other hand, the management of software and tools used in computer science is what represents today a fundamental pillar in what teaching and scientific research refers, however, the concept goes far beyond that, of the hand of computer security that involves many more factors than are believed, not only physical but logical, referring to all the applications that run on each of the computers and that are part of the software. This paper presents a secure software model based on the software development life cycle and aims to create professionals capable of developing secure software products and at the same time raise awareness of the various problems that may arise from not establishing security measures in Systems development A final analysis of the importance of creating professionals with knowledge of the use and application of computer security is carried out.

Keywords: Software engineering; informatic security; systems security; cybercrimes; security tools.

Introducción

La ingeniería en software busca ordenar y maximizar la eficiencia de la creación de software dentro de las empresas. Según García-Peñalvo (García-Peñalvo, 2018), la ingeniería en software nace por: "La incapacidad de las organizaciones para predecir tiempo, esfuerzos y costes en el desarrollo de software producido son dos de las principales bases sobre las que surge la Ingeniería del Software como una disciplina científica".

Dentro del ciclo de vida del software se puede entender por seguridad los requerimientos del sistema y poder cumplir con sus funciones, pero realmente no es así, se estima que hay una gran confusión con el tema seguridad el cual es un concepto delicado con respecto al manejo de datos. Es importante aplicar la seguridad al ciclo de vida del software con el fin de tener calidad en los productos desarrollados y cumplir con los tres objetivos de la seguridad que son la integridad, confidencialidad y disponibilidad (Garzón, 2010).

Por otro lado, si estos objetivos no son acatados correctamente por los ingenieros de software, sus sistemas no podrán continuar su desarrollo del ciclo de vida ya que la seguridad introduce no sólo características de calidad, sino también restricciones bajo las cuales el sistema debe operar. Ignorar tales restricciones durante el proceso de desarrollo podría llevar a serios problemas (David G. Rosado).

De la misma manera, la seguridad informática debe ir de la mano junto a la ingeniería de software para mantener el desarrollo del sistema en óptimas condiciones junto a los requerimientos de la evolución de la tecnología y las necesidades a satisfacer de usuarios y proveedores a los sistemas que estos son empleados.

En el desarrollo de software la información enfrenta riesgos de daño o pérdida. Como resultado a la creciente demanda de la interconexión masiva y global, los sistemas y las redes de información se han vuelto más vulnerables ya que están expuestos a una cantidad creciente, así como a una mayor variedad de amenazas. Esto hace a su vez que surjan nuevos retos que deben abordarse en materia de seguridad. La seguridad informática pretende eliminar o contener estos daños o pérdidas (Voutssas, 2010).

El modelo propuesto busca desarrollar las habilidades y crear conciencia a los estudiantes que quizás aún no tienen la experiencia en el desarrollo de software o ciberseguridad y se espera que se evalúen todas las vulnerabilidades a las que podrían estar expuestos los programas desarrollados. Además, mediante este modelo se busca determinar cuál es el nivel de la evaluación de los programas para determinar e identificar las posibles vulnerabilidades o debilidades del código que sean fáciles de explotar aumentando así el conocimiento y el uso de la ciberseguridad.

Marco teórico

Ingeniería de software

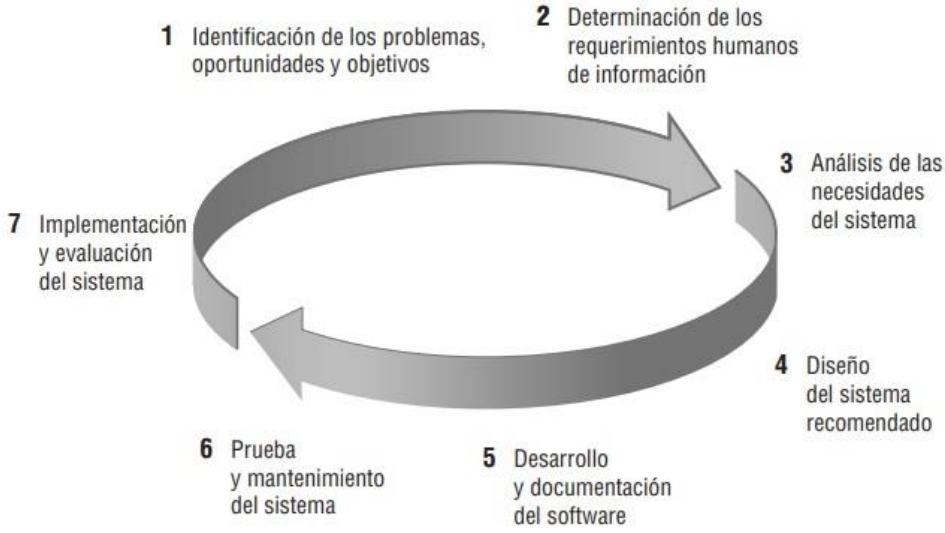
Como se ha mencionado previamente la ingeniería en software lo que busca es poder ordenar y maximizar la eficiencia de la creación de software dentro de las empresas. Según (García-Peñalvo, 2018), la ingeniería en software nace por: "La incapacidad de las organizaciones para predecir tiempo, esfuerzos y costes en el desarrollo de software producido son dos de las principales bases sobre las que surge la Ingeniería del Software como una disciplina científica". El manejo que se tiene del Software y las herramientas utilizadas en informática es lo que representa hoy en día un pilar fundamental en lo que enseñanza e investigación científica se refiere.

Sommerville, define a la ingeniería de software como una disciplina que abarca etapas desde la inicialización de un sistema hasta la post-instalación y puesta en marcha que incluye el mantenimiento (Sommerville, 2005).

El concepto de Ingeniería de software va mucho más allá, esta disciplina surge de la distinción que debe realizarse entre el desarrollo de pequeños proyectos y el desarrollo de grandes proyectos. El SDLC (ciclo de desarrollo de sistemas) es una metodología en fases para el análisis y diseño, de acuerdo con la cual los sistemas se desarrollan mejor al utilizar un ciclo específico de actividades del analista y los usuarios (Kendall y Kendall, 2011). En la figura 1.1 se muestra las siete fases del ciclo de desarrollo de sistemas (SDLC) según Kendall y Kendall.

Figura 1

Ciclo de desarrollo de sistemas



Fuente: (Kendall y Kendall, 2011)

Seguridad informática

Voutssas, define que la seguridad informática es: el proceso de establecer y observar un conjunto de estrategias, políticas, técnicas, reglas, guías, prácticas y procedimientos tendientes a prevenir, proteger y resguardar de daño, alteración o sustracción a los recursos informáticos de una organización y que administren el riesgo al garantizar en la mayor medida posible el correcto funcionamiento ininterrumpido de esos recursos (Voutssas, 2010).

"Common Vulnerabilities and Exposures" (CVE, 2019) es una lista de vulnerabilidades y exposiciones de ciberseguridad pública, reconocida ampliamente por la comunidad internacional de ciberseguridad, según CVE, una vulnerabilidad es "una debilidad en la lógica computacional (por ejemplo, el código) que se encuentra en el software y algunos componentes de hardware (por ejemplo, el firmware) que, cuando se explota, produce un impacto negativo en la confidencialidad, la integridad o la disponibilidad. La mitigación de las vulnerabilidades en este contexto implica cambios en la codificación, pero también podría incluir cambios en las especificaciones o incluso la eliminación de las especificaciones (por ejemplo, la eliminación de los protocolos o la funcionalidad afectados en su totalidad)".

Ingeniería de software y seguridad informática

Desde el punto de vista de software, los datos son un bien que debe ser protegido desde todos los aspectos. En el tema de seguridad existen varias guías o estándares, sin embargo, uno de los más conocidos y que deberían utilizar todos los ingenieros de software es OWASP, una comunidad de seguridad informática sin fines de lucro que trabaja para crear artículos, metodologías, documentación, herramientas y tecnologías que pueden ser usadas gratuitamente.

Existe un top 10 de los principales riesgos de seguridad según OWASP que tiene un alto impacto relacionado con las vulnerabilidades de aplicaciones web más vistas y por ende en la seguridad orientada al desarrollo de software (Ruiz, 2018). Los cambios se han acelerado en los últimos cuatro años, y OWASP Top 10 necesitaba actualizarse (Foundation, 2017). En la figura 2 se muestra los cambios del top 10 de OWASP del 2013 al 2017 donde se ha mejorado la metodología, utilizado un nuevo proceso de obtención de datos, OWASP ha trabajado con la comunidad, reordenando los riesgos y reescribiéndolos desde cero, además, se ha agregado referencias a frameworks y lenguajes más utilizados en la actualidad.

Figura 2

OWASP Top 10 – 2017 Los diez riesgos más críticos en Aplicaciones Web

OWASP Top 10 2013	±	OWASP Top 10 2017
A1 – Inyección	→	A1:2017 – Inyección
A2 – Pérdida de Autenticación y Gestión de Sesiones	→	A2:2017 – Pérdida de Autenticación y Gestión de Sesiones
A3 – Secuencia de Comandos en Sitios Cruzados (XSS)	↘	A3:2017 – Exposición de Datos Sensibles
A4 – Referencia Directa Insegura a Objetos [Unido+A7]	∪	A4:2017 – Entidad Externa de XML (XXE) [NUEVO]
A5 – Configuración de Seguridad Incorrecta	↘	A5:2017 – Pérdida de Control de Acceso [Unido]
A6 – Exposición de Datos Sensibles	↗	A6:2017 – Configuración de Seguridad Incorrecta
A7 – Ausencia de Control de Acceso a las Funciones [Unido+A4]	∪	A7:2017 – Secuencia de Comandos en Sitios Cruzados (XSS)
A8 – Falsificación de Peticiones en Sitios Cruzados (CSRF)	☒	A8:2017 – Deserialización Insegura [NUEVO, Comunidad]
A9 – Uso de Componentes con Vulnerabilidades Conocidas	→	A9:2017 – Uso de Componentes con Vulnerabilidades Conocidas
A10 – Redirecciones y reenvíos no validados	☒	A10:2017 – Registro y Monitoreo Insuficientes [NUEVO, Comunidad]

Fuente: (Foundation, 2017)

Los ingenieros de software enfocan su trabajo en realizar un adecuado análisis y diseño de aplicaciones de software; sin embargo, en ciertas ocasiones dejan pasar por alto aspectos y recomendaciones de seguridad dados por OWASP, los mismos que se deben implementar a nivel de diseño, codificación o implementación y así reducir el riesgo de amenaza permanente y directa sobre este tipo de aplicaciones o servicios donde el punto vital de ellos es demostrar que la seguridad de la información y el software puede operar junto para poder tener un producto de software que tengan los más grandes estándares de calidad, esto se obtiene realizando las pruebas de seguridad que se le debería realizar a todos nuestros productos donde esté involucrado el software (Guamán, Guamán, Jaramillo, y Sucunuta, 2017).

El uso de las metodologías agiles para la seguridad informática

Como ya vimos la seguridad informática es un tema de suma importancia, ya que la información se ha convertido en algo tan valioso que hay que tener diferentes mecanismos de defensa para que los datos no sean expuestos a terceros. Los cibercriminales desarrollan nuevas técnicas para hacerse con la información, a la par de que la tecnología avanza surgen nuevas brechas de seguridad lo que provoca realizar constantes actualizaciones ya sea a hardware o software, los equipos de desarrollo tienen el trabajo de estar en constante estudio de las nuevas técnicas de robo de información, es ahí donde entran las metodologías agiles, el desarrollo de un software que permite estar en constante cambio para adaptarse a los nuevos riesgos, al hacer uso de estas metodologías permite que el desarrollador cree un software que puede ser entregado por ejemplo en prototipos y que en cada prototipo de que se entrega y logran burlar la seguridad el desarrollador tendrá la capacidad de sellar dicha brecha al seguir desarrollando el programa y así en el nuevo prototipo ya no se veré afectado por el ataque anterior.

El desarrollo ágil y la computación en la nube son un complemento ideal. Según Cole, con la computación en la nube, implementar nuevos servidores y llamar a nuevos servicios administrados es rápido, lo que permite a los desarrolladores y equipos iterar rápidamente. La computación en la nube ofrece a los desarrolladores acceso rápido y bajo demanda a una variedad de entornos de prueba diferentes. La computación en la nube y el paradigma de infraestructura como código permiten a los desarrolladores configurar e implementar firewalls, redes lógicas y mecanismos de autenticación / autorización de manera declarativa. Esto permite a los desarrolladores centrarse en la seguridad de la misma manera que el hardware y el software, y empuja la seguridad a una posición central en el proceso de desarrollo y operaciones de las aplicaciones en la nube. La computación en la nube también permite pruebas de seguridad automatizadas, un componente importante del desarrollo ágil de software (Brian S. Cole, 2018). El paradigma ágil mejora la productividad en muchos sentidos y la computación en la nube ofrece muchas vías para el desarrollo ágil. Además, los mecanismos de cifrado y los registros de acceso para almacenar datos basados en la nube ofrecen capacidades de auditoría específicas para que los investigadores demuestren su cumplimiento.

Metodo para el desarrollo de software seguro basado en la ciberseguridad e ingeniería de software

El método que se propone para el aseguramiento de software surge gracias a un esfuerzo conjunto del personal docente de la Facultad de Ingenierías de una universidad ecuatoriana ubicada en el cantón Samborondón y los estudiantes de la carrera de ingeniería en Sistemas de primer semestre. Durante la puesta en marcha de este experimento surgieron necesidades e identificación de escenarios reales que los estudiantes enfrentan en el desarrollo de sus proyectos de software.

Este método consta de tres etapas: comprensión de conceptos básicos de ingeniería de software y ciberseguridad, administración y control de vulnerabilidades y evaluación del método para el desarrollo de software seguro y la toma de decisiones.

Comprensión de conceptos básicos de ingeniería de software y ciberseguridad

Las metodologías actuales, sugieren que la seguridad no es una característica para ampliar la funcionalidad del software. En este estudio enfatizamos que el problema de la seguridad de software empieza en la fase del desarrollo, lo que se propone es hacer un enfoque global en el diseño de un software seguro cuyo desarrollo se realice en varias fases.

En la primera etapa de la investigación se centra en comprender los conceptos básicos relacionados a la ingeniería de software y la seguridad informática. En esta segunda etapa, el modelo propuesto se basa en el estándar OWASP y sus métricas donde se puede categorizar en que vulnerabilidades se encuentra susceptible el software desarrollado. Además, para robustecer el modelo, se analiza cada una de las etapas del análisis de seguridad del software cuyo objetivo es crear software seguro durante la implementación y no después de haber sido desarrollado, con esto se busca crear profesionales capacitados en el ámbito no solo de programación sino de programación segura y libre de vulnerabilidades.

Métricas OWASP

La tabla 1 muestra las métricas según el top 10 de OWASP para calcular la probabilidad y medidas de prevención posibles (Foundation, 2017)

Tabla 1

Evaluación de vulnerabilidades y prevención de riesgos en seguridad de aplicaciones

Riesgo	¿La aplicación es vulnerable?	Prevención
Inyección	• Los datos suministrados por el usuario no son validados, filtrados o sanitizados por la aplicación. • Se invocan consultas dinámicas o no parametrizadas, sin codificar los parámetros de forma acorde al contexto. • Se utilizan datos dañinos dentro de los parámetros de búsqueda en consultas Object-Relational Mapping (ORM), para extraer registros adicionales sensibles. • Los datos dañinos se usan directamente o se concatenan, de modo que el SQL o comando resultante contiene datos y estructuras con consultas dinámicas, comandos o procedimientos almacenados.	La opción preferida es utilizar una API segura, que evite el uso de un intérprete por completo y proporcione una interfaz parametrizada. Se debe migrar y utilizar una herramienta de Mapeo Relacional de Objetos (ORMs). • Realice validaciones de entradas de datos en el servidor, utilizando "listas blancas". De todos modos, esto no es una defensa completa ya que muchas aplicaciones requieren el uso de caracteres especiales, como en campos de texto, APIs o aplicaciones móviles. • Para cualquier consulta dinámica residual, escape caracteres especiales utilizando la sintaxis de caracteres específica para el intérprete que se trate. • Utilice LIMIT y otros controles SQL dentro de las consultas para evitar la fuga masiva de registros en caso de inyección SQL.
Pérdida de Autenticación	• Permite ataques automatizados como la reutilización de credenciales conocidas, cuando el atacante ya posee una lista de pares de usuario y contraseña válidos. • Permite ataques de fuerza bruta y/o ataques automatizados. • Permite contraseñas por defecto, débiles o muy conocidas, como “Password1”, “Contraseña1” o “admin/admin”. • Posee procesos débiles o inefectivos en el proceso de	• Implemente autenticación multi-factor para evitar ataques automatizados, de fuerza bruta o reúso de credenciales robadas. • No utilice credenciales por defecto en su software, particularmente en el caso de administradores. • Implemente controles contra contraseñas débiles. Cuando el usuario ingrese una nueva clave, la misma puede verificarse contra la lista del Top 10.000 de peores contraseñas.

Riesgo	¿La aplicación es vulnerable?	Prevención
	recuperación de credenciales, como "respuestas basadas en el conocimiento", las cuales no se pueden implementar de forma segura. • Almacena las contraseñas en textos claros o cifrados con métodos de hashing débiles (vea A3:2017-Exposición de Datos Sensibles). • No posee autenticación multi-factor o fue implementada de forma ineficaz. • Expone Session IDs en las URL, no la invalida correctamente o no la rota satisfactoriamente luego del cierre de sesión o de un periodo de tiempo determinado.	• Alinear la política de longitud, complejidad y rotación de contraseñas con las recomendaciones de la Sección 5.1.1 para Secretos Memorizados de la Guía NIST 800-63 B's u otras políticas de contraseñas modernas, basadas en evidencias. • Mediante la utilización de los mensajes genéricos iguales en todas las salidas, asegúrese que el registro, la recuperación de credenciales y el uso de APIs, no permiten ataques de enumeración de usuarios. • Limite o incremente el tiempo de respuesta de cada intento fallido de inicio de sesión. Registre todos los fallos y avise a los administradores cuando se detecten ataques de fuerza bruta. • Utilice un gestor de sesión en el servidor, integrado, seguro y que genere un nuevo ID de sesión aleatorio con alta entropía después del inicio de sesión. El Session-ID no debe incluirse en la URL, debe almacenarse de forma segura y ser invalidado después del cierre de sesión o de un tiempo de inactividad determinado por la criticidad del negocio
Exposición de Datos Sensibles	• ¿Se transmite datos en texto claro? Esto se refiere a protocolos como HTTP, SMTP, TELNET, FTP. El tráfico en Internet es especialmente peligroso. Verifique también todo el tráfico interno, por ejemplo, entre los balanceadores de carga, servidores web o sistemas de backend. • ¿Se utilizan algoritmos criptográficos obsoletos o débiles, ya sea por defecto o en código heredado? Por ejemplo MD5, SHA1, etc.	Clasifique los datos procesados, almacenados o transmitidos por el sistema. Identifique qué información es sensible de acuerdo a las regulaciones, leyes o requisitos del negocio y del país. • Aplique los controles adecuados para cada clasificación. • No almacene datos sensibles innecesariamente. Descártelos tan pronto como sea posible o utilice un sistema de tokenizacion que cumpla con PCI DSS. Los datos que no se almacenan no pueden ser robados.

Riesgo	¿La aplicación es vulnerable?	Prevención
	• ¿Se utilizan claves criptográficas predeterminadas, se generan o reutilizan claves criptográficas débiles, o falta una gestión o rotación adecuada de las claves? • Por defecto, ¿se aplica cifrado? ¿se han establecido las directivas de seguridad o encabezados para el navegador web? • ¿El User-Agent del usuario (aplicación o cliente de correo), verifica que el certificado enviado por el servidor sea válido?	• Cifre todos los datos sensibles cuando sean almacenados. • Cifre todos los datos en tránsito utilizando protocolos seguros como TLS con cifradores que utilicen Perfect Forward Secrecy (PFS), priorizando los algoritmos en el servidor. Aplique el cifrado utilizando directivas como HTTP Strict Transport Security (HSTS). • Utilice únicamente algoritmos y protocolos estándares y fuertes e implemente una gestión adecuada de claves. No cree sus propios algoritmos de cifrado. • Deshabilite el almacenamiento en cache de datos sensibles. • Almacene contraseñas utilizando funciones de hashing adaptables con un factor de trabajo (retraso) además de SALT, como Argon2, scrypt, bcrypt o PBKDF2. • Verifique la efectividad de sus configuraciones y parámetros de forma independiente.
Entidades Externas XML (XXE)	• La aplicación acepta XML directamente, carga XML desde fuentes no confiables o inserta datos no confiables en documentos XML. Por último, estos datos son analizados sintácticamente por un procesador XML. • Cualquiera de los procesadores XML utilizados en la aplicación o los servicios web basados en SOAP, poseen habilitadas las definiciones de tipo de documento (DTDs). Dado que los mecanismos exactos para deshabilitar el procesamiento de DTDs varía para cada procesador, se recomienda consultar la hoja de trucos para prevención de XXE de OWASP. • La aplicación utiliza SAML para el procesamiento de identidades	• De ser posible, utilice formatos de datos menos complejos como JSON y evite la serialización de datos confidenciales. • Actualice los procesadores y bibliotecas XML que utilice la aplicación o el sistema subyacente. Utilice validadores de dependencias. Actualice SOAP a la versión 1.2 o superior. • Deshabilite las entidades externas de XML y procesamiento DTD en todos los analizadores sintácticos XML en su aplicación, según se indica en la hoja de trucos para prevención de XXE de OWASP. • Implemente validación de entrada positiva en el servidor (“lista blanca”), filtrado y sanitización para prevenir el

Riesgo	¿La aplicación es vulnerable?	Prevención
	dentro de la seguridad federada o para propósitos de Single Sign-On (SSO). SAML utiliza XML para garantizar la identidad de los usuarios y puede ser vulnerable. • La aplicación utiliza SOAP en una versión previa a la 1.2 y, si las entidades XML son pasadas a la infraestructura SOAP, probablemente sea susceptible a ataques XXE. • Ser vulnerable a ataques XXE significa que probablemente la aplicación también es vulnerable a ataques de denegación de servicio, incluyendo el ataque Billion Laughs.	ingreso de datos dañinos dentro de documentos, cabeceras y nodos XML. • Verifique que la funcionalidad de carga de archivos XML o XSL valide el XML entrante, usando validación XSD o similar. • Las herramientas SAST pueden ayudar a detectar XXE en el código fuente, aunque la revisión manual de código es la mejor alternativa en aplicaciones grandes y complejas. • Si estos controles no son posibles, considere usar parcheo virtual, gateways de seguridad de API, o Firewalls de Aplicaciones Web (WAFs) para detectar, monitorear y bloquear ataques XXE.
Pérdida de Control de Acceso	• Pasar por alto las comprobaciones de control de acceso modificando la URL, el estado interno de la aplicación o HTML, utilizando una herramienta de ataque o una conexión vía API. • Permitir que la clave primaria se cambie a la de otro usuario, pudiendo ver o editar la cuenta de otra persona. • Elevación de privilegios. Actuar como un usuario sin iniciar sesión, o actuar como un administrador habiendo iniciado sesión como usuario estándar. • Manipulación de metadatos, como reproducir un token de control de acceso JWT (JSON Web Token), manipular una cookie o un campo oculto para elevar los privilegios, o abusar de la invalidación de tokens JWT. • La configuración incorrecta de CORS permite el acceso no autorizado a una API. • Forzar la navegación a páginas autenticadas como un usuario no	• Con la excepción de los recursos públicos, la política debe ser denegar de forma predeterminada. • Implemente los mecanismos de control de acceso una vez y reutilícelo en toda la aplicación, incluyendo minimizar el control de acceso HTTP (CORS). • Los controles de acceso al modelo deben imponer la propiedad (dueño) de los registros, en lugar de aceptar que el usuario puede crear, leer, actualizar o eliminar cualquier registro. • Los modelos de dominio deben hacer cumplir los requisitos exclusivos de los límites de negocio de las aplicaciones. • Deshabilite el listado de directorios del servidor web y asegúrese que los metadatos/fuentes de archivos (por ejemplo de GIT) y copia de seguridad no estén presentes en las carpetas públicas. • Registre errores de control de acceso y alerte a los administradores cuando corresponda (por ej. fallas reiteradas).

Riesgo	¿La aplicación es vulnerable?	Prevención
	autenticado o a páginas privilegiadas como usuario estándar. • Acceder a una API sin control de acceso mediante el uso de verbos POST, PUT y DELETE.	• Limite la tasa de acceso a las APIs para minimizar el daño de herramientas de ataque automatizadas. • Los tokens JWT deben ser invalidados luego de la finalización de la sesión por parte del usuario. • Los desarrolladores y el personal de QA deben incluir pruebas de control de acceso en sus pruebas unitarias y de integración.
Configuración de Seguridad Incorrecta	• Falta hardening adecuado en cualquier parte del stack tecnológico, o permisos mal configurados en los servicios de la nube. • Se encuentran instaladas o habilitadas características innecesarias (ej. puertos, servicios, páginas, cuentas o permisos). • Las cuentas predeterminadas y sus contraseñas siguen activas y sin cambios. • El manejo de errores revela a los usuarios trazas de la aplicación u otros mensajes demasiado informativos. • Para los sistemas actualizados, las nuevas funciones de seguridad se encuentran desactivadas o no se encuentran configuradas de forma adecuada o segura. • Las configuraciones de seguridad en el servidor de aplicaciones, en el framework de aplicación (ej., Struts, Spring, ASP.NET), bibliotecas o bases de datos no se encuentran especificados con valores seguros. • El servidor no envía directrices o cabeceras de seguridad a los clientes o se encuentran configurados con valores inseguros. • El software se encuentra desactualizado o posee vulnerabilidades.	• Proceso de fortalecimiento reproducible que agilice y facilite la implementación de otro entorno asegurado. Los entornos de desarrollo, de control de calidad (QA) y de Producción deben configurarse de manera idéntica y con diferentes credenciales para cada entorno. Este proceso puede automatizarse para minimizar el esfuerzo requerido para configurar cada nuevo entorno seguro. • Use una plataforma minimalista sin funcionalidades innecesarias, componentes, documentación o ejemplos. Elimine o no instale frameworks y funcionalidades no utilizadas. • Siga un proceso para revisar y actualizar las configuraciones apropiadas de acuerdo a las advertencias de seguridad y siga un proceso de gestión de parches. En particular, revise los permisos de almacenamiento en la nube (por ejemplo, los permisos de buckets S3). • La aplicación debe tener una arquitectura segmentada que proporcione una separación efectiva y segura entre componentes y acceso a terceros, contenedores o grupos de seguridad en la nube (ACLs). • Envíe directivas de seguridad a los clientes (por ej. Cabeceras de seguridad).

Riesgo	¿La aplicación es vulnerable?	Prevención
		• Utilice un proceso automatizado para verificar la efectividad de los ajustes y configuraciones en todos los ambientes.
Cross-Site Scripting (XSS)	XSS Reflejado: la aplicación o API utiliza datos sin validar, suministrados por un usuario y codificados como parte del HTML o Javascript de salida. No existe una cabecera que establezca la Política de Seguridad de Contenido (CSP). Un ataque exitoso permite al atacante ejecutar comandos arbitrarios (HTML y Javascript) en el navegador de la víctima. Típicamente el usuario deberá interactuar con un enlace, o alguna otra página controlada por el atacante, como un ataque del tipo pozo de agua, publicidad maliciosa, o similar. • XSS Almacenado: la aplicación o API almacena datos proporcionados por el usuario sin validar ni sanear, los que posteriormente son visualizados o utilizados por otro usuario o un administrador. Usualmente es considerado como de riesgo de nivel alto o crítico. • XSS Basados en DOM: frameworks en JavaScript, aplicaciones de página única o APIs incluyen datos dinámicamente, controlables por un atacante. Idealmente, se debe evitar procesar datos controlables por el atacante en APIs no seguras.	• Utilizar frameworks seguros que, por diseño, automáticamente codifican el contenido para prevenir XSS, como en Ruby 3.0 o React JS. • Codificar los datos de requerimientos HTTP no confiables en los campos de salida HTML (cuerpo, atributos, JavaScript, CSS, o URL) resuelve los XSS Reflejado y XSS Almacenado. La hoja de trucos OWASP para evitar XSS tiene detalles de las técnicas de codificación de datos requeridas. • Aplicar codificación sensitiva al contexto, cuando se modifica el documento en el navegador del cliente, ayuda a prevenir DOM XSS. Cuando esta técnica no se puede aplicar, se pueden usar técnicas similares de codificación, como se explica en la hoja de trucos OWASP para evitar XSS DOM. • Habilitar una Política de Seguridad de Contenido (CSP) es una defensa profunda para la mitigación de vulnerabilidades XSS, asumiendo que no hay otras vulnerabilidades que permitan colocar código malicioso vía inclusión de archivos locales, bibliotecas vulnerables en fuentes conocidas almacenadas en Redes de Distribución de Contenidos (CDN) o localmente.
Deserialización Insegura	Aplicaciones y APIs serán vulnerables si deserializan objetos hostiles o manipulados por un atacante.	• Implemente verificaciones de integridad tales como firmas digitales en cualquier objeto serializado, con el fin de detectar modificaciones no autorizadas. • Durante la deserialización y antes de la creación del objeto, exija el

Riesgo	¿La aplicación es vulnerable?	Prevención
		cumplimiento estricto de verificaciones de tipo de dato, ya que el código normalmente espera un conjunto de clases definibles. Se ha demostrado que se puede pasar por alto esta técnica, por lo que no es aconsejable confiar sólo en ella. • Aísle el código que realiza la deserialización, de modo que se ejecute en un entorno con los mínimos privilegios posibles. • Registre las excepciones y fallas en la deserialización, tales como cuando el tipo recibido no es el esperado, o la deserialización produce algún tipo de error. • Restrinja y monitoree las conexiones (I/O) de red desde contenedores o servidores que utilizan funcionalidades de deserialización. • Monitoree los procesos de deserialización, alertando si un usuario deserializa constantemente.
Uso de Componentes con Vulnerabilidades Conocidas	• No conoce las versiones de todos los componentes que utiliza (tanto del lado del cliente como del servidor). Esto incluye componentes utilizados directamente como sus dependencias anidadas. • El software es vulnerable, no posee soporte o se encuentra desactualizado. Esto incluye el sistema operativo, servidor web o de aplicaciones, DBMS, APIs y todos los componentes, ambientes de ejecución y bibliotecas. • No se analizan los componentes periódicamente ni se realiza seguimiento de los boletines de seguridad de los componentes utilizados. • No se parchea o actualiza la plataforma subyacente, frameworks y dependencias, con un enfoque	• Remover dependencias, funcionalidades, componentes, archivos y documentación innecesaria y no utilizada. • Utilizar una herramienta para mantener un inventario de versiones de componentes (por ej. frameworks o bibliotecas) tanto del cliente como del servidor. Por ejemplo, Dependency Check y retire.js. • Monitorizar continuamente fuentes como CVE y NVD en búsqueda de vulnerabilidades en los componentes utilizados. Utilizar herramientas de análisis automatizados. Suscribirse a alertas de seguridad de los componentes utilizados. • Obtener componentes únicamente de orígenes oficiales utilizando canales seguros. Utilizar preferentemente

Riesgo	¿La aplicación es vulnerable?	Prevención
	basado en riesgos. Esto sucede comúnmente en ambientes en los cuales la aplicación de parches se realiza de forma mensual o trimestral bajo control de cambios, lo que deja a la organización abierta innecesariamente a varios días o meses de exposición a vulnerabilidades ya solucionadas. • No asegura la configuración de los componentes correctamente	paquetes firmados con el fin de reducir las probabilidades de uso de versiones manipuladas maliciosamente. • Supervisar bibliotecas y componentes que no poseen mantenimiento o no liberan parches de seguridad para sus versiones obsoletas o sin soporte. Si el parcheo no es posible, considere desplegar un parche virtual para monitorizar, detectar o protegerse contra la debilidad detectada.
Registro y Monitoreo Insuficientes	• Eventos auditables, tales como los inicios de sesión, fallos en el inicio de sesión, y transacciones de alto valor no son registrados. • Advertencias y errores generan registros poco claros, inadecuados o ninguno en absoluto. • Registros en aplicaciones o APIs no son monitoreados para detectar actividades sospechosas. • Los registros son almacenados únicamente de forma local. • Los umbrales de alerta y de escalamiento de respuesta no están implementados o no son eficaces. • Las pruebas de penetración y escaneos utilizando herramientas DAST (como OWASP ZAP) no generan alertas. • La aplicación no logra detectar, escalar o alertar sobre ataques en tiempo real.	• Asegúrese de que todos los errores de inicio de sesión, de control de acceso y de validación de entradas de datos del lado del servidor se pueden registrar para identificar cuentas sospechosas. Mantenerlo durante el tiempo suficiente para permitir un eventual análisis forense. • Asegúrese de que las transacciones de alto impacto tengan una pista de auditoría con controles de integridad para prevenir alteraciones o eliminaciones. • Asegúrese que todas las transacciones de alto valor poseen una traza de auditoría con controles de integridad que permitan detectar su modificación o borrado, tales como una base de datos con permisos de inserción únicamente u similar. • Establezca una monitorización y alerta efectivos de tal manera que las actividades sospechosas sean detectadas y respondidas dentro de períodos de tiempo aceptables. • Establezca o adopte un plan de respuesta o recuperación de incidentes, tales como NIST 800-61 rev.2 o posterior.

Fuente: OWASP Top 10 - 2017 Los diez riesgos más críticos en Aplicaciones Web, Foundation, O. (2017)

Análisis de seguridad del software

La tabla 2 muestra la propuesta para la distribución procentual del esfuerzo realizado en cada una de las etapas del ciclo de vida de desarrollo del software.

Tabla 2

Distribución porcentual de cada etapa del ciclo de vida de software

ETAPA	Valor % mínimo	Valor % máximo	Valor % medio
Análisis	1	12	3
Diseño	15	41	24
Codificación	21	50	38
Pruebas	15	33	22
Validación	3	22	8
Mantenimiento	1	11	5

Fuente: Elaboración propia

Análisis de software

Se identifican los requerimientos funcionales que tendrán impacto en relación con los aspectos de seguridad de la aplicación.

Diseño de software

En esta etapa hay varios aspectos de seguridad que deben ser contemplados, entre ellos se puede citar los siguientes diseños:

- Autorización
- Autenticación
- Mensajes de error
- Mecanismos de protección de datos

Codificación de software

Esta etapa es muy importante y a la vez delicada ya que suele introducirse vulnerabilidades muchas veces por error o falta de análisis.

Pruebas de software

En este punto el objetivo es encontrar y reportar errores funcionales de la aplicación desarrollada.

Validación de software

En esta etapa un conjunto de procesos de análisis y comprobación aseguran que el software que se desarrolla vaya acorde a su especificación y cumple las necesidades de los clientes.

Mantenimiento y evaluación de software

Esta fase involucra cambios al software para corregir defectos y dependencias encontradas durante su uso tanto como la adición de nueva funcionalidad para mejorar la usabilidad y aplicabilidad del software.

A continuación, se muestran los tipos de mantenimientos existentes:

Perfectivo: Mejora del software (rendimiento, flexibilidad, reusabilidad) o implementación de nuevos requisitos. También se conoce como mantenimiento evolutivo.

Adaptativo: Adaptación del software a cambios en su entorno tecnológico (nuevo hardware, otro sistema de gestión de bases de datos, otro sistema operativo)

Correctivo: Corrección de fallos detectados durante la explotación.

Preventivo: Facilitar el mantenimiento futuro del sistema (verificar precondiciones, mejorar legibilidad).

Resultados

El modelo propuesto actualmente está en una etapa de experimentación por lo cual aún no tenemos resultados definitivos. El curso seleccionado de estudiantes de la materia ingeniería de software se encuentra en el primer mes de clases, participan alrededor de 30 estudiantes.

Se planea llevar una estrategia de investigación mixta que combine estudios tanto cualitativos como cuantitativos para la recolección de evidencia. Actualmente se está desarrollando actividades para mejorar la enseñanza y junto con un estudio de encuesta cualitativa para evaluar la eficacia del modelo seguro de software. Se planea elaborar encuestas y usar múltiples métodos de recolección de datos como entrevistas, análisis de documentos y cuestionarios con el objetivo de diseñar teorías a partir de estudios de casos de investigación. Las entrevistas serán validadas por medio de entrevistas piloto con grupos de personas para cada muestra.

Se ha seccionado dos grupos de participantes:

1. Estudiantes del curso de ingeniería de software para obtener información sobre la percepción al usar este modelo.
2. Personal docente involucrado en el proyecto para recopilar datos sobre el contexto.

Conclusiones

Este artículo ha presentado un modelo para la seguridad de software basado en las métricas de OWASP y en el ciclo de vida de software. Se identificó como tendencia en el desarrollo de software que el tema de seguridad no se trata suficientemente y por ende surge la necesidad de combinar estrategias activas y proactivas con el fin de disminuir los riesgos asociados a las vulnerabilidades existentes en los sistemas desarrollados. La propuesta combina el enfoque en el ciclo de vida del software, seguridad informática y análisis de riesgos que permita a los estudiantes desarrollar habilidades de prevención con el uso de herramientas y metodologías. La experimentación del modelo se encuentra en progreso, uno de los objetivos del mismo es permitir a otros cursos y universidades el uso de un modelo que permita contribuir al aseguramiento del producto final de software y al crecimiento de la experiencia y colaboración del tema de seguridad informática aplicado al desarrollo de software.

Referencias Bibliográficas

Brian S. Cole, J. H. (2018). Eleven quick tips for architecting biomedical informatics workflows with cloud computing. *PLOS Computational Biology Education*.

CVE. (2019). *Common Vulnerabilities and Exposures*. Obtenido de cve.mitre.org/

David G. Rosado, C. B.-M. (s.f.). La Seguridad como una asignatura indispensable. *XVI Jornadas de Enseñanza Universitaria de la Informática*.

Foundation, O. (2017). *OWASP Top 10 - 2017 Los diez riesgos más críticos en Aplicaciones Web.* Obtenido de www.owasp.org

García-Peñalvo, F. J. (2018). Obtenido de https://repositorio.grial.eu/bitstream/grial/1228/1/07-rep.pdf

Garzón, P. A. (2010). *DragonJar*. Obtenido de https://www.dragonjar.org/seguridad-informatica-un-reto-para-la-ingenieria-del-software-o-una-necesidad.xhtml

Guamán, D., Guamán, F., Jaramillo, D., y Sucunuta, M. (2017). Implementation of techniques and OWASP security recommendations to avoid SQL and XSS attacks using J2EE and WS-Security. *2017 12th Iberian Conference on Information Systems and Technologies (CISTI)*.

Kendall, K., y Kendall, J. (2011). *Análisis y diseño de sistemas.* México: Pearson Educación.

Mathkour H., A. G. (2008). A Risk Management Tool for Extreme Programming. *IJCSNS International Journal of Computer Science and Network Security, 8*(8), 326-333.

Ruiz, G. (9 de 10 de 2018). *Los 10 Principales Riesgos de Seguridad según OWASP – Parte I*. Obtenido de https://blog.sucuri.net/espanol/2018/10/los-10-principales-riesgos-de-seguridad-segun-owasp-parte-i.html

Schnoeller G., M. L. (2016). A strategy based on knowledge acquisition for management of requirements risks on distributed XP development. *Revista lbérica de Sistemas y Tecnologías de Información*(20), 18–33. doi:10.17013/risti.20.18–33

SEI. (2010). *CMMI para Desarrollo, Versión 1.3. Mejora de los procesos para el desarrollo de mejores productos y servicios.* EE.UU.: Technical Report, Software Engineering Institute.

SGI. (2014). *The CHAOS Manifesto.* Obtenido de The Standish Group International: https://www.projectsmart.co.uk/white-papers/chaos-report.pdf

SGI. (2015). *Standish Group 2015 Chaos Report.* Obtenido de https://www.infoq.com/articles/standish-chaos-2015

Sommerville, I. (2005). *Ingeniería del Software.* Pearson Educación.

Vondran, A. (2015). *Metodologías ágiles de gestión de proyectos.* Obtenido de https://www.linkedin.com/pulse/metodolog%C3%ADas-%C3%A1giles-de-gesti%C3%B3n-proyectos-andre-vondran/

Voutssas, J. (2010). Preservación documental digital y seguridad informática. *Investigación bibliotecológica*.