2. Seguridad en el Ciclo de Vida del Software en las Fases de Análisis y Diseño

2.1. Introducción y objetivos

Actualmente, el aumento de los ataques al software vulnerable ha dejado patente la insuficiencia de las protecciones a nivel de infraestructura. En este contexto, es conveniente minimizar al máximo los ataques en la capa de aplicación y, por lo tanto, el número de vulnerabilidades explotables.

El desarrollo de software seguro y confiable requiere de la adopción de un proceso sistemático o disciplina que aborde la seguridad en cada una de las fases de su ciclo de vida. Se debe integrar en él dos tipos de actividades de seguridad: la primera es el seguimiento de unos principios de diseño seguro (mínimo privilegio, por ejemplo); y la segunda, la inclusión de una serie de buenas prácticas de seguridad(especificación de requisitos de seguridad, casos de abuso, análisis de riesgo, análisis de código, pruebas de penetración dinámicas, etc.). A este nuevo ciclo de vida con prácticas de seguridad incluidas lo llamaremos S-SDLC.

Una de las principales ventajas de la adopción de un S-SDLC es el descubrimiento de errores de codificación y debilidades de diseño en las etapas tempranas de su desarrollo, lo que implica un importante ahorro de costes.

A diagram of firewall and firewall

Description automatically generated

Figura 1. Ataques de la capa aplicación. Fuente: Hoglund, 2004.

Normalmente, el modelo más propicio para el desarrollo de software en las organizaciones suele ser una combinación de dos o más modelos de los ciclos de vida (cascada, espiral, etc.); sin embargo, lo importante no es el modelo seguido, si no la incorporación de buenas prácticas de seguridad en las diferentes fases de este, y unos hitos o puntos de control donde se verifiquen los entregables de las mismas.

A lo largo del tema se presentará un conjunto mínimo de prácticas de seguridad que deberían ser una parte integral del S-SDLC. Entre las razones principales para añadir prácticas de seguridad en el SDLC, tenemos:

* Mayor probabilidad de capturar adecuadamente los requisitos, tomar decisiones de diseño correctas y no cometer errores involuntarios de codificación.
* Dificultad de los agentes maliciosos para explotar vulnerabilidades y debilidades del software, pues el resultante de un ciclo de vida S-SDLC será más robusto en su entorno de ejecución y en su interacción con entidades externas.

Los objetivos del presente tema son los siguientes:

* Conocimiento y comprensión de las buenas prácticas de seguridad a incluir en un S-SDLC en las fases de requisitos y diseño.
* Profundizar en el estudio de una de las principales prácticas de seguridad, como es el análisis de riesgo arquitectónico.
* Introducir al alumno en prácticas de seguridad con el modelado de amenazas, los casos de abuso, el modelado de ataques, la ingeniería de requisitos de seguridad y los patrones de diseño.

2.2. La seguridad en ciclo de vida del software

La seguridad del software es algo más que la eliminación de las vulnerabilidades y la realización de pruebas de penetración. Un aspecto importante es la adopción por los gerentes de un enfoque sistémico que incorpore los principios de diseño y unas buenas prácticas de seguridad del *software touchpoint* en su ciclo de vida de desarrollo. Su objetivo es producir software más seguro y confiable, así como el poder verificar su seguridad. A este nuevo ciclo de vida lo denominaremos ciclo de vida del software seguro S-SDLC.

No existe una metodología de ingeniería de seguridad de software que haya probado unos mejores resultados que otras, todas se basan en la inclusión de prácticas de seguridad, fundamentalmente. Por lo tanto, no importa cuál es el modelo de ciclo de vida seguido, lo que sí es importante es que la seguridad sea considerada desde las primeras etapas del ciclo de vida del software.

A diagram of a diagram of a company

Description automatically generated

Figura 2. Mejores prácticas de seguridad del software en el SDLC. Fuente: elaboración propia.

La figura anterior propone un Ciclo de Vida de Desarrollo de Software SDLC en cascada, basado en el modelo de McGraw (2005). Para otro tipo, como los iterativos, sería similar, donde se especifican las actividades y prácticas de seguridad a efectuar en cada fase de este. A continuación, se enumeran esas actividades o mejores prácticas de seguridad del software:

* Modelado de Amenazas
* Casos de abuso.
* Modelado de ataques.
* Ingeniería requisitos de seguridad.
* Análisis de riesgo arquitectónico.
* Patrones de diseño.
* Pruebas de seguridad basadas en riesgo.
* Revisión de código.
* Pruebas de penetración.
* Operaciones de seguridad.
* Revisión externa.

En la siguiente tabla se representan las prácticas de seguridad y su relación en cada una de las fases donde son aplicables.

A blue and white table with text

Description automatically generated with medium confidence

Tabla 1. Relación entre las prácticas de seguridad y las fases del ciclo de vida. Fuente: elaboración propia.

En los siguientes apartados de este tema, se desarrollarán las prácticas de seguridad propuestas para el ciclo de vida.

2.3. Modelado de amenazas

Una amenaza para una aplicación de sistema software es cualquier actor, agente, circunstancia o evento que tiene el potencial de causarle daño a los datos o recursos a los que tiene, o permite, acceso (no se debe confundir con el concepto de vulnerabilidad).

A diagram of a diagram of a company

Description automatically generated

Figura 3. Modelado de amenazas. Fuente: elaboración propia.

Las amenazas se pueden clasificar según su intencionalidad:

* Involuntaria.
* Intencional, pero no malicioso.
* Maliciosa.

Esta importante practica de seguridad se desarrolla en los siguientes vídeos:

En este vídeo *(Modelado de Amenazas)* se presenta y estudia la buena práctica de seguridad Modelado de Amenazas, a desarrollar en las fases de análisis y diseño de un ciclo de vida de desarrollo de un software (SDLC).

En este vídeo *(Microsoft Threat Modelling Tool)*se realiza una demostración práctica de la herramienta de modelado de amenazas para aplicaciones Microsoft Threat Modelling Tool.

### **2.4. Casos de abuso**

Los **diagramas de casos de uso** constituyen unas buenas prácticas para la obtención de los requisitos funcionales de una aplicación; sin embargo, para la obtención de requisitos de seguridad no lo son, sobre todo los**no funcionales o requisitos negativos** referidos a actividades que no debería realizar el sistema. Para solucionar lo anterior, se ha creado un tipo especializado de casos de uso que se utiliza para analizar y especificar las amenazas de seguridad. Xiaohong y sus colaboradores (2015) lo definen como:

«Un caso de abuso es la inversa de un caso de uso, es decir, una función que el sistema no debe permitir o una secuencia completa de acciones que resulta en una pérdida para la organización» (Xiaohong et al., 2015).

A diagram of a diagram of a company

Description automatically generated

Figura 4. Casos de abuso. Fuente: elaboración propia.

Los **casos de abuso,** o casos de mal uso, son un instrumento que puede ayudar a pensar de la misma forma que lo hacen los atacantes. Pensando más allá de los aspectos normativos, funcionales y estudiando los eventos negativos o inesperados. Los **profesionales de seguridad de** **software** entienden mejor cómo crear software seguro, pues permiten obtener una mejor comprensión de las áreas de riesgo del sistema a través de:

* Identificación de los **objetivos de seguridad** que debe cumplir el software.
* Identificación de las **amenazas de seguridad** a ser mitigadas por el software.
* Identificación de los puntos en el software **susceptibles a ser atacados.**
* Obtener las restricciones o requisitos no funcionales de seguridad, requisitos negativos, necesarios para alcanzar los **objetivos de seguridad.**
* Obtener **requisitos de seguridad** funcionales, requisitos positivos, de los mecanismos que garanticen que el software cumple con los objetivos de seguridad.

Los casos de abuso constituyen un excelente medio de análisis de las amenazas que debe afrontar el software. Son apropiados para el análisis y especificación de restricciones, o requisitos negativos, ya que se basan en el uso indebido del sistema. Establecen la base para otros casos de uso de seguridad que proporcionan los medios para contrarrestar o mitigar las amenazas capturadas en los mismos y una manera altamente reutilizable de organizar, analizar y especificar sus requisitos de seguridad.

Los casos de abuso describen lo que el software**no debe hacer** en respuesta al uso incorrecto o malintencionado de este por un agente malicioso. Para cada caso de uso funcional, el desarrollador deberá explorar las formas en que esa función podría ser deliberadamente mal utilizada. También tiene que identificar todas las posibles relacionesentre casos de uso de seguridad y casos abuso, para especificar restricciones y requisitos de seguridad para evitar un mal uso o abuso de las funcionalidades del software.

En la siguiente figura se muestra la relación entre el caso de **uso de seguridad** y el de**abuso asociado.**

A diagram of a security system

Description automatically generated

Figura 5. Relación entre el caso de uso de seguridad y el de abuso asociado. Fuente: adaptado de Donald, 2003.

En la siguiente tabla se resumen las diferencias entre los casos de uso de seguridad y los casos de abuso:

A table with text on it

Description automatically generated

Tabla 2. Diferencias entre los casos de uso de seguridad y los casos de abuso. Fuente: adaptado de Donald, 2003.

A continuación, presentamos un ejemplo de un diagrama de casos de uso de seguridad y sus casos de abuso asociados en su versión gráfica. La siguiente figura (Sindre y Opdahl, 2001) muestra un caso de abuso para un caso de comercio electrónico que presenta varias relaciones:

* **Relación caso abuso:** uso seguridad, «incluye» y «extiende».
* **Relación caso de uso:** abuso, «previene» y «detecta».

Los **casos uso** resultantes de especificar los requisitos de seguridad protegen al cajero automático y a sus usuarios de las amenazas potencialmente realizables por un cibercriminal.

![A diagram of a diagram

Description automatically generated]()

Figura 6. Ejemplo caso de uso de seguridad y caso de abuso. Fuente: adaptado de Sindre y Opdahl, 2001.

La siguiente tabla expone un ejemplo de especificación del caso de abuso **obtener contraseña** del diagrama de casos de uso y abuso de la figura anterior.

A screenshot of a blue and white list

Description automatically generated

Tabla 3. Especificación caso de abuso Obtener Contraseña. Fuente: elaboración propia.

### **2.5. Modelado de ataques**

El principal **objetivo** de la seguridad del software es el **mantener sus propiedades de seguridad** frente a los ataques realizados por personal malicioso sobre sus componentes y reducir al mínimo posible sus vulnerabilidades explotables. Para conseguir que el desarrollo de una aplicación posea las propiedades y los principios de diseño del software seguro, se necesita que el personal de diseño y desarrollo generen dos perspectivas:

A screenshot of a computer program

Description automatically generated

Figura 7. Perspectivas de modelado. Fuente: elaboración propia.

La **perspectiva del atacante** se suele modelar de las siguientes dos formas:

* Patrones de ataque.
* Árboles de ataque.

A diagram of a diagram of a company

Description automatically generated

Figura 8. Modelado de ataques. Fuente: elaboración propia.

El uso combinado de patrones de ataque y de árboles de ataque captura la probabilidad de **cómo los ataques se pueden combinar y secuenciar,** esto proporciona una serie de datos que nos permiten ayudar a diseñar, en el software, una serie de respuestas encaminadas a mitigarlos.

### **Patrones de ataque**

Un ataque aprovecha una**vulnerabilidad** de una aplicación mediante un exploitpara obtener un beneficio del sistema como escalada de privilegios, robo y modificación de datos, modificaciones del funcionamiento, denegación de servicio, etc.

Un **patrón de ataque** se pude definir como:

Mecanismo o medio para capturar y representar la perspectiva y el conocimiento del ciberatacante con el suficiente detalle acerca de cómo los ataques llevan a cabo los métodos más frecuentes de explotación (exploit) y las técnicas usadas para comprometer el software.

En definitiva, constituyen una clasificación de los ataques yuna **representación estructurada** del pensamiento del atacante para que el equipo de diseño y desarrollo obtenga el conocimiento y los pasos necesarios a realizar para mitigar con mayor probabilidad las acciones de los ciberatacantes, reducir su impacto y seleccionar las políticas de seguridad más convenientes. Derivan del concepto de **patrones de diseño** aplicado en un contexto más destructivo que constructivo y están generados a partir de un análisis en profundidad de determinados ejemplos de exploits del mundo real.

Un catálogo de patrones de ataques, que proporciona un conjunto de definiciones comunes, una taxonomía de clasificación, un esquema de patrones de ataque y un conjunto de ellos, reales, lo constituye la iniciativa del MITRE **Common Attack Pattern Enumeration and Classification (CAPEC).** Actualmente, incluye 519 patrones reales de ataque.

Accede al catálogo a través del aula virtual o desde la siguiente dirección web: <http://capec.mitre.org>

Dependiendo del nivel de detalle que describe y su nivel de abstracción, los patrones de ataque constituyen un recurso que proporciona**valor** al conjunto del SDLC, dado que tiene diferentes utilidades en cada una de sus fases

En la siguiente tabla se muestra un resumen de los usos de los patrones de ataques en las diferentes fases del SDLC.

A screenshot of a computer program

Description automatically generated

Tabla 4. Aplicación patrones de ataque a las diferentes fases del SDLC. Fuente: elaboración propia.

Un **patrón de ataque,** como mínimo, debe describir ampliamente el incidente, las habilidades y recursos necesarios para ejecutarlo con éxito, en qué contexto es aplicable y proporcionar información suficiente para permitir a los defensores evitar o mitigar eficazmente las acciones del atacante.

En la tabla siguiente se muestran los ítems de información de los que consta un patrón de ataque obtenido de CAPEC.

A screenshot of a computer

Description automatically generated

A table of informational text

Description automatically generated with medium confidence

Tabla 5. Alcance de información de un patrón de ataque. Fuente: elaboración propia

### **Árboles de ataque**

Un árbol de ataque se puede definir como un método sistemático para caracterizar la seguridad de un sistema, basado en la combinación y dependencias de las vulnerabilidades de este, que un atacante puede aprovechar para comprometerlo.

En un árbol de ataque, el **objetivo del atacante** se coloca en la parte superior del árbol, documentándose las posibles alternativas de ataque en los diferentes recorridos del árbol por los nodos de nivel inferior que contienen objetivos intermedios hasta llegar al nivel inferior que contiene los diferentes métodos o **técnicas de ataque.** Cada camino a través de un árbol de ataque representa un tipo de ataque único. Para cada alternativa se puede añadir recursivamente otras precursoras que alcancen diversos objetivos parciales que, en conjunto, logran el objetivo principal del atacante.

Mediante el examen de diferentes ramas del árbol de ataque, el analista puede identificar **todas las posibles técnicas o métodos** que podrían ser utilizados para comprometer la seguridad del sistema. Básicamente:

* Captura los **pasos realizados** en ataques con éxito.
* Captura **métodos generales y específicos** del sistema de ataque, propiedades del sistema y otras condiciones previas que lo hacen posible.
* Se enfoca en las **causas de las vulnerabilidades,** pero no identifica las contramedidas.
* Representan de forma **gráfica y textual** directrices de codificación segura y patrones de seguridad.

La representación de la **estructura de un árbol de ataque** se realiza conforme a los dos siguientes tipos:

* **Textual:** sigue una estructura de esquema numérico, el nodo raíz, o la meta, representada en el **primer nivel** sin sangría y cada objetivo de nivel inferior se enumeran con sangría de una unidad por nivel de descomposición:

![A white text on a white background

Description automatically generated]()

Figura 9. Estructura textual de un árbol de ataque. Fuente: elaboración propia.

* **Grafica o semántica:** se construye, generalmente, con el nodo raíz, o la meta, en la **parte superior,** al descender por las ramas del árbol, se obtienen subobjetivos hasta que se llega al nivel inferior donde están los métodos de ataque. Un nodo se descompone como:

A blue and white rectangular boxes with text

Description automatically generated with medium confidence

Figura 10. Aplicación patrones de ataque a las diferentes fases del SDLC. Fuente: elaboración propia.

A diagram of a computer network

Description automatically generated

Figura 11. Vista conceptual de un árbol de ataque. Fuente: adaptado de Redwine, S. T. Jr., 2006.

### **2.6. Ingeniería de requisitos de seguridad**

Gran parte de las vulnerabilidades y debilidades del software tienen su origen en unos requisitos inadecuados, inexactos e incompletos, principalmente debido a la falta o debilidad de la especificación de estos, que no determinan las funciones, restricciones y propiedades no funcionales del software que hacen que este sea previsible, confiable y resistente.

Los defectos en los requisitos cuestan de 10 a 200 veces más de corregir durante la **ejecución** que si se detectan durante su **especificación.** Además, es difícil y costoso mejorar significativamente la seguridad de una aplicación después de que esté en su entorno de producción.

La **fase de ingeniería** de requisitos cubre todas las actividades y tareas que deben realizarse antes de iniciar el diseño, su principal resultado es la especificación de los requisitos que definen los aspectos funcionales y no funcionales del software.

A diagram of a diagram of a company

Description automatically generated

Figura 12. Ingeniería de requisitos. Fuente: elaboración propia.

Los requisitos de seguridad deben ser considerados simultáneamente junto con los demás requisitos, incluidos los relativos a funcionalidad, rendimiento y facilidad de uso. Ya que los requisitos de seguridad, a menudo, entran en conflicto con ellos, se hace necesario, por lo tanto, la integración satisfactoria de la ingeniería de requisitos de seguridad con la ingeniería de requisitos en su conjunto.

En la siguiente figura se muestra un alcance completo de los requisitos de seguridad de una aplicación:

A diagram of a software security system

Description automatically generated

Figura 13. Requisitos de seguridad. Fuente: elaboración propia.

* **Requisitos servicios de seguridad (funcionales o positivos).** Incluye la especificación de funciones que implementan una política de seguridad, como control de acceso, autenticación, autorización, cifrado y gestión de claves. Deben especificar:
  + **Propiedades que el** **software debe exhibir,** por ejemplo: el software debe tener un comportamiento correcto y predecible y disponer de capacidades de recuperación frente a ciberataques.
  + **Nivel requerido de seguridad y salvaguardas de riesgo** de las funciones de seguridad.
* **Requisitos de** **software** **seguro (no funcionales o negativos).** Requisitos que afectan directamente a la probabilidad de que el software sea seguro. Estos abarcan, principalmente, los no funcionales, los que garanticen que el sistema seguirá siendo confiable, incluso cuando esa confianza se vea amenazada.
* **Operacionales.** Mas enfocados a los controles y normas que rigen los procesos de desarrollo, implementación y operación del software.

Es fundamental que los **requisitos de seguridad** del software sean:

* Completos.
* Precisos.
* Coherentes.
* Trazables.
* Verificables.

Teniendo en cuenta las prácticas de seguridad anteriormente presentadas, los pasos a realizar para la especificación de los requisitos de seguridad se explican a continuación:

* Dibujar **DFD** para el sistema que implementa los casos de uso.
* Identificar las **amenazas potenciales** para cada elemento del DFD mediante la metodología de modelado de amenazas.
* Identificar los**casos de uso** basados en las amenazas identificadas. Solo se identifican los nombres de los casos de abuso y el objetivo de cada caso de abuso.
* **Mapear las amenazas** obtenidas con los patrones de ataque de CAPEC, así como otras fuentes de patrones de ataque. Proporcionará información para la especificación de los caos de uso y valoración del riesgo de las amenazas
* **Usar los patrones de ataque recuperados** para especificar los casos de abuso identificados.
* Especificar los requisitos de seguridad funcionales en base a las **salvaguardas de seguridad** necesarias para mitigar las amenazas; y los no funcionales, en base a los patrones de ataque y caso de buso.

A diagram of a company

Description automatically generated

Figura 14. Proceso de especificación de requisitos de seguridad. Fuente: elaboración propia.

### **2.7. Análisis de riesgo arquitectónico**

Según estudios y estadísticas disponibles, se sabe que, aproximadamente, un 50 % de los defectos de seguridad se deben a errores en la fase de diseño denominados **debilidades** oflaws, en el idioma inglés.

Identificando el riesgo, y después con su gestión, se pueden mitigar gran parte de los problemas de seguridad de un software.

A diagram of a diagram of a company

Description automatically generated

Figura 15. Análisis de riesgos. Fuente: elaboración propia.

En la figura anterior se puede observar que el análisis de riesgos está a caballo entre la**fase análisis de requisitos,** donde se obtienen los requisitos de seguridad, se modelan ataques y realizan casos de abuso, y la fase de la **fase de arquitectura y diseño.** Sigue en importancia a la revisión de código, si bien este hecho puede variar dependiendo de las características de la organización, de sus tipos de sistemas, cantidad de software, etc.

En la fase de análisis, al obtener los requisitos de seguridad se han llegado a bastantes conclusiones acerca de los activos del sistema y del impacto que los posibles ataques pueden causar, pero es en la **fase de diseño de la arquitectura del sistema** donde se especifican todos los activos, porque se obtienen todos los componentes hardware y software del sistema, se configura la arquitectura y se deciden las salvaguardas concretas que van a protegerlo en función de los requisitos de seguridad y los casos de abuso.

### **Proceso análisis de riesgo arquitectónico**

**Cigital** usa el **proceso de análisis de riesgo arquitectónico** que desarrolla un acercamiento a un análisis de riesgo que implica tres pasos básicos.

Modelo de Cigital, Building Security In:

A diagram of a company

Description automatically generated

Figura 16. Fases análisis de riesgo arquitectónico. Fuente: elaboración propia.

* **Análisis de resistencia al ataque.** Realiza un modelado de amenazas del diseño completo de la aplicación para analizar la resistencia al ataque. Para ello comprueba una **lista de categorías de riesgos** según el modelo de análisis de amenazas de Microsoft STRIDE.
* **Análisis de ambigüedad.** Es un subproceso que pretende descubrir nuevos riesgos en base al conocimiento de principios de diseño. Aprovecha múltiples puntos de vista de la **arquitectura de** **software**, de varios arquitectos, para crear una **técnica de análisis** **crítica** con el fin de encontrar nuevos defectos, puntos de conflicto y de inconsistencia.
* **Análisis de debilidad.** Es un subproceso que ayuda al entendimiento del impacto de dependencias del software externo.

### **2.8. Patrones de diseño**

Según se define en el documento de referencias (McGraw, 2005), los patrones de diseño son:

«Una solución general repetible a un problema de ingeniería de software recurrente, que está expresamente destinado a contribuir al diseño de software menos vulnerable, más resistente y tolerante a los ataques» (McGraw, 2005).

A diagram of a diagram of a company

Description automatically generated

Figura 17. Patrones de diseño. Fuente: elaboración propia.

El concepto de **patrones** es más familiar en **ingeniería de** **software,** relacionados principalmente con la arquitectura de los sistemas o soluciones para el diseño de estos. Se han desarrollado patrones de diseño de seguridad para casi **todas las fases del SDLC:** requerimientos, análisis, diseño, codificación, etc. El uso adecuado de estos patrones conduce a la **remediación** de los principales **fallos de seguridad,** como, por ejemplo, uno que tuvo un efecto significativo fue un **validador** que construía un modelo de validación de entrada para defenderse contra los ataques de inyección SQL; otros ejemplos son los construidos para los **lenguajes de programación** en**C y /** o **C + +.**