Seguridad en el ciclo de vida del *software*

[2.1] ¿Cómo estudiar este tema?

[2.2] Introducción a la seguridad en ciclo de vida del *software* (S- SDLC)

[2.3] Seguridad en las fases del S-SDLC

[2.4]Modelado de ataques

[2.5] Casos de abuso

[2.6] Ingeniería de requisitos de seguridad

[2.7] Análisis de riesgo. Arquitectónico

[2.8] Patrones de diseño

[2.9] Pruebas de seguridad basadas en riesgo

[2.10] Revisión de código

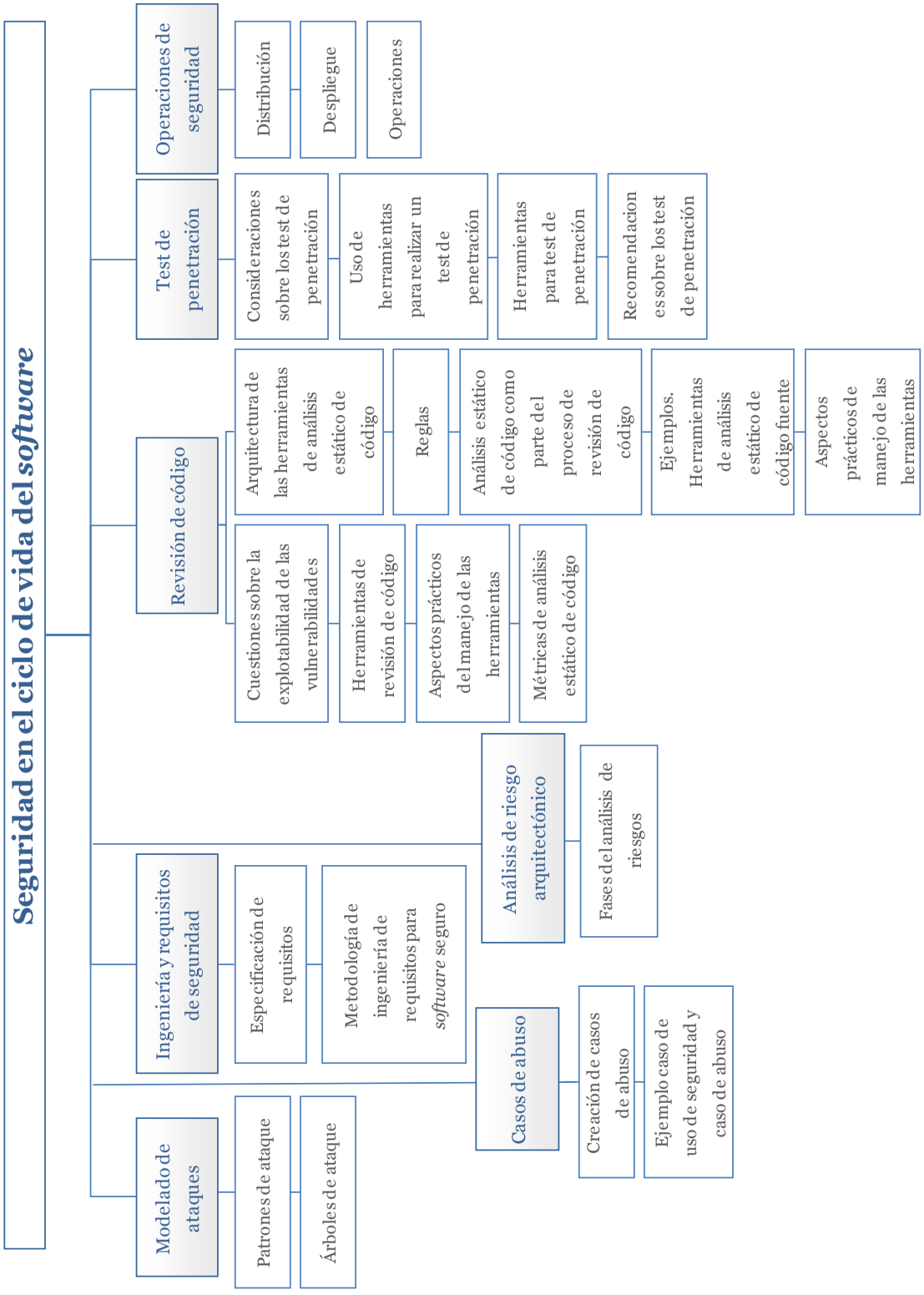
[2.11] Test de penetración

[2.12] Operaciones de seguridad

[2.13] Revisión externa

TEMA 2

Esquema



Ideas clave

2.1. ¿Cómo estudiar este tema?

Para estudiar este tema debes leer las ideas clave y de los apuntes elaborados por el profesor «Tema 2: Seguridad en el ciclo de vida del *software* ».

El objetivo del presente tema es introducir al alumno en las diferentes **prácticas de seguridad** a introducir en el SDLC adoptado por una organización al objeto de obtener *software* más seguro, confiable, que presente un mínimo de vulnerabilidades y sea resistente a los ataques provenientes del entorno interior como del exterior.

2.2. Introducción a la seguridad en ciclo de vida del *software* (S-SDLC)

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 el **minimizar al máximo los ataques en la capa de aplicación y por tanto en el número de vulnerabilidades explotables**.

El desarrollo de *software* seguro y confiable requiere la adopción de un proceso sistemático o disciplina que aborde la seguridad **en cada una de sus fases de su ciclo de vida** e integre actividades de seguridad como el seguimiento de unos **principios de diseño seguro** (mínimo privilegio, etc.) y la inclusión de una **serie de buenas prácticas de seguridad** (especificación requisitos 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**.

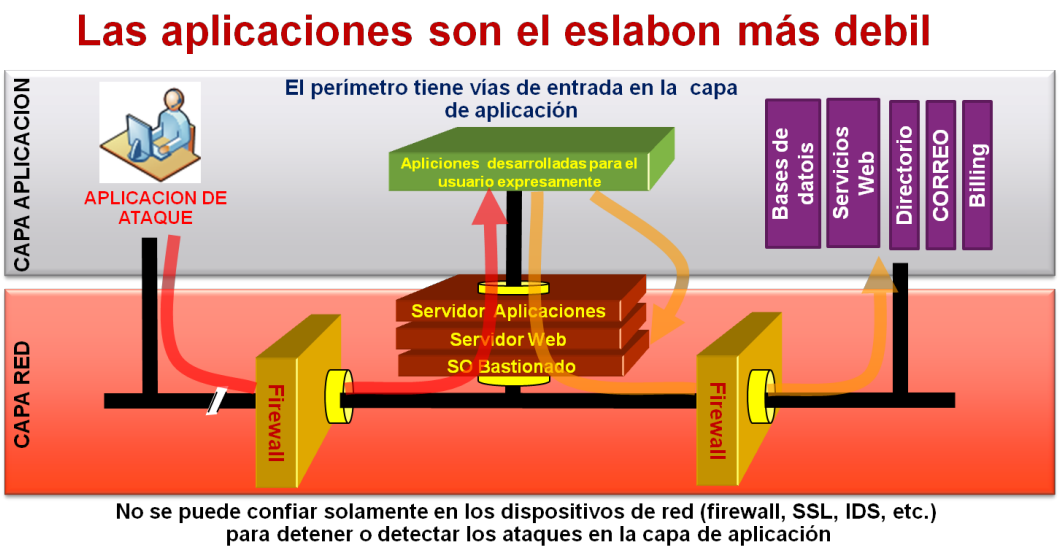


Figura 1. Ataques capa aplicación. Adaptada de Hoglund, G. and McGraw, G. (2004). *Exploiting Software: How to Break Code*. Addison-Wesley Software Security Series

2.3. Seguridad en las fases del S-SDLC

La **seguridad del *software*** es algo más que la eliminación de las vulnerabilidades y la realización de pruebas de penetración, **es una disciplina**. Un aspecto importante de la misma, tal y como ya se ha comentado en el párrafo anterior, es la adopción de un enfoque sistémico para **incorporar buenas prácticas de seguridad del *software* «*touchpoint*»** en cada fase del SDLC **y unos hitos de control**.

Tal y como se ha comentado en el tema anterior, 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 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***.

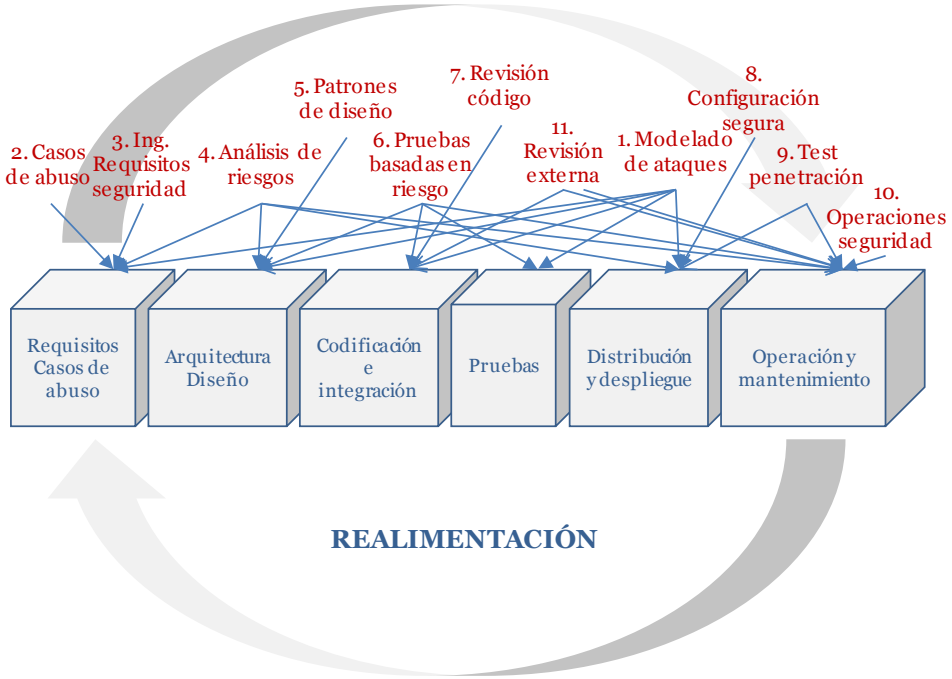


Figura 2. Mejores prácticas de seguridad del *software* en el SDLC. Adaptada de Hoglund, G. and McGraw, G. (2004). *Exploiting Software: How to Break Code*. Addison-Wesley Software Security Series

La figura anterior, muestra un ejemplo de ciclo de vida de desarrollo de *software* SDLC en cascada, para otro tipo como el iterativo sería similar, donde se especifican las actividades y pruebas de seguridad a efectuar en cada fase del mismo. A continuación se enumeran esas actividades o mejores prácticas de seguridad del *software* tomando como referencia el modelo MCGraw, al que se le añaden otras:

- Modelado de ataques.
- Casos de abuso.
- Ingeniería requisitos de seguridad.
- Análisis de riesgo arquitectónico.
- Patrones de diseño.
- Pruebas de seguridad basadas en riesgo.
- Revisión de código.
- Test penetración.
- Configuraciones seguras.
- Operaciones de Seguridad.
- Revisión externa.

2.4. Modelado de ataques

El principal objetivo de la seguridad del *software* es el de **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 principios de diseño del *software* seguro presentadas en el tema 1, se necesita que el personal de diseño y desarrollo desarrollen dos perspectivas:

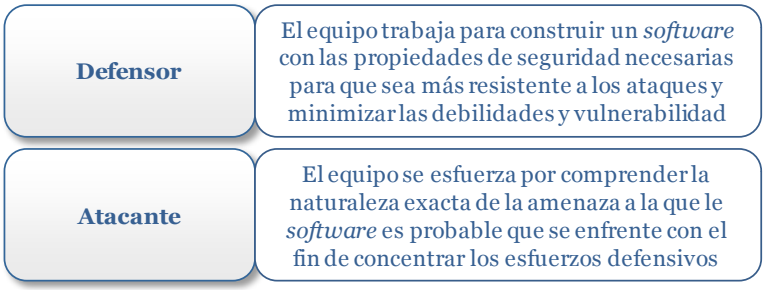


Figura 3. Perspectivas de modelado

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

- **Patrones de ataque**. Constituyen un mecanismo o medio para capturar y representar la perspectiva y conocimiento del ciberatacante con el suficiente detalle acerca de cómo los ataques se llevan a cabo, los métodos más frecuentes de explotación (*exploit*) y las técnicas usadas para comprometer el *software*.

- **Árboles de ataque**. Un método sistemático para caracterizar la seguridad de un sistema, basado en la combinación y dependencias de las vulnerabilidades del mismo, que un atacante puede aprovechar para comprometerlo.

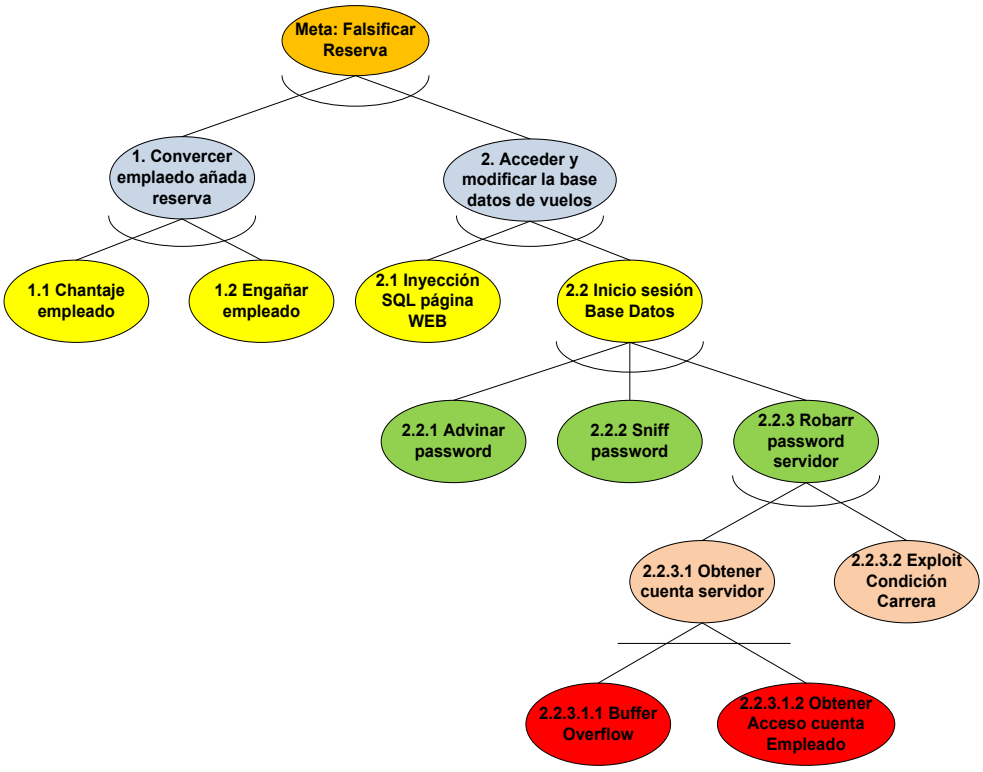


Figura 4. Vista conceptual de un árbol de ataque

2.5. Casos de abuso

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.

Pensando más allá de los aspectos normativos y funcionales y también estudiando 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:

- Identificar los objetivos de seguridad que debe cumplir el *software*.
- Identificación de las amenazas de seguridad a ser neutralizadas por el *software*.
- Identificación de los puntos en el *software* susceptibles de ser atacados.
- Definición de restricciones, o requisitos negativos, necesarios para alcanzar las contramedidas necesarias y los objetivos de seguridad.

- Obtener requisitos de seguridad que garanticen que el *software* aplica las restricciones necesarias.

Los casos de abuso describen lo que el *software* no debe hacer en respuesta al **uso incorrecto o malintencionado** del mismo por un agente malicioso. Para cada caso de uso funcional, el desarrollador debería explorar las formas en que esa función podría ser deliberadamente mal utilizada.

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

	Casos de abuso	Casos usos seguridad
Uso	Analiza y especifica la amenazas a la seguridad	Analiza y especifica los requisitos de seguridad
Criterio de éxito	Éxito del atacante	Éxito de la aplicación
Producido	Equipo de seguridad	Equipo de seguridad
Usado	Equipo de seguridad	Equipo de requisitos
Actor externo	Atacantes y usuarios	Usuarios
Conducido por	Análisis de vulnerabilidades de activos y amenazas	Casos de abuso

Tabla 1 Diferencias entre los casos de uso de seguridad y los casos de abuso.
Extraída de Donald, G. (2003). Security Use Cases. *Journal of Object Technology*. Software Engineering Institute, U.S.A

El método más simple y práctico para crear casos de abuso es por lo general mediante un **proceso de *brainstorming* o tormenta de ideas.** El proceso de creación de casos de abuso, consta de tres etapas bien diferenciadas, tal y como se muestra en un esquema simplificado del proceso de modelar los casos de abuso en la figura siguiente:

- **Documentación de entrada**.
- **Actividades**.
- **Documentación de salida**.

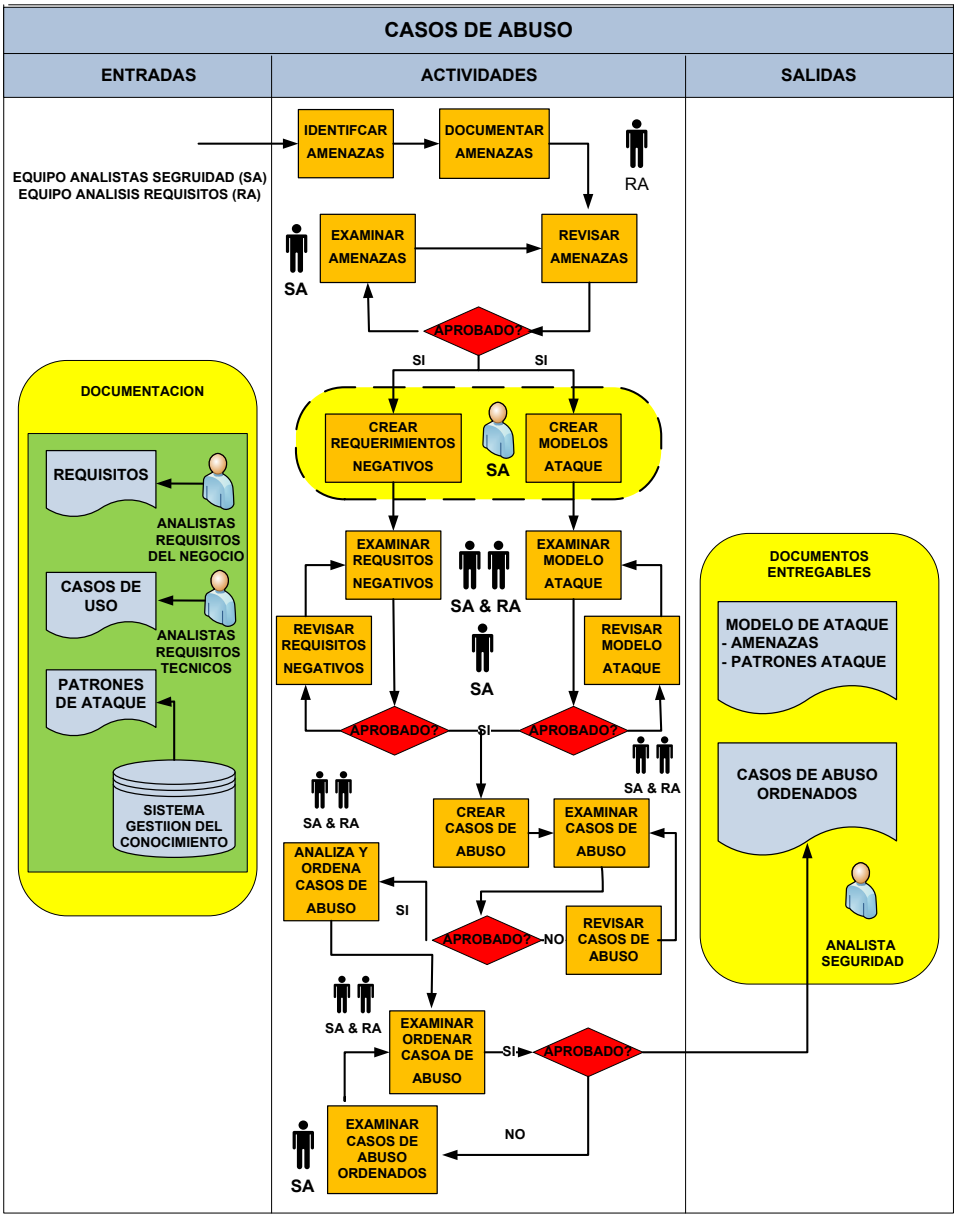


Figura 5. Proceso de creación de casos de abuso

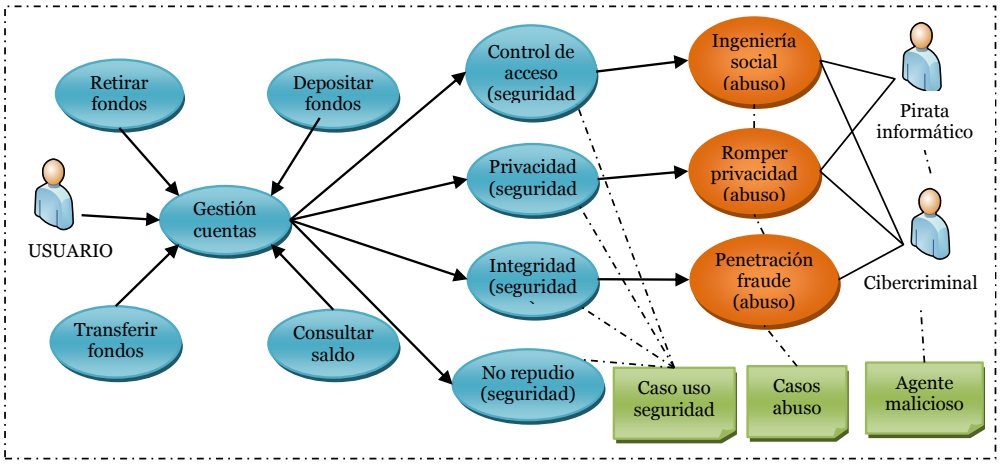


Figura 6. Ejemplo caso de uso de seguridad y caso de abuso

2.6. Ingeniería de requisitos de seguridad

Gran parte de la mayoría 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 los mismos que no determinan las funciones, restricciones y propiedades no funcionales del *software* que hacen que este sea previsible, confiable y resistente.

Los **requisitos de las funcionalidades o servicios de seguridad del sistema o el *software* a menudo se confunden con los requisitos de *software* seguro**:

- **Requisitos servicios de seguridad**. Incluye la especificación de funciones que implementan una política de seguridad, como control de acceso, autenticación, autorización, cifrado y gestión de claves. Estas funciones previenen la violación de las propiedades de seguridad del sistema o de la información que procesa, como el acceso no autorizado, modificación, denegación de servicio, etc.

- **Requisitos de *software* seguro**. 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. Estos requisitos se dirigen hacia la reducción o eliminación de las vulnerabilidades del *software*, como validación de entradas, manejo de excepciones, ejecución entornos aislados etc.

La figura siguiente muestra una vista de alto nivel de las tareas y artefactos involucrados en la fase de requisitos del ciclo de vida del *software*.

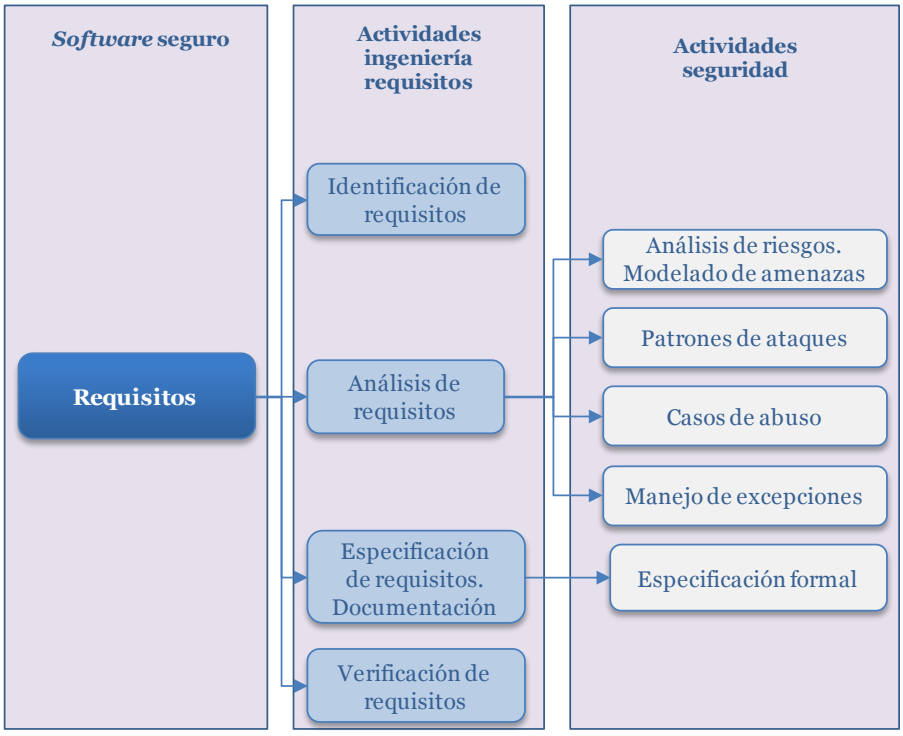


Figura 7. Vista de alto nivel de las tareas y artefactos involucrados en la fase de requisitos

Un modelo útil simplificado de una aplicación para la especificación y análisis de requisitos, es el representado en la figura siguiente, en el que de forma general se incluyen cinco componentes que contienen todos los programas.

Figura 8. Componentes generales de una aplicación. Extraída de Redwine, S. T. Jr. (Editor) (2006). *Software Assurance: A Guide to the Common Body of Knowledge to Produce, Acquire, and Sustain Secure Software Version 1.1.* US Department of Homeland Security

- **Interfaz de entrada**.
- **Interfaz de salida**.
- **Datos internos**.
- **Datos críticos de seguridad** como claves criptográficas, datos relacionados con privilegios y otros datos críticos.
- **Algoritmos**. Lógica interna del programa, que procesa los datos internos.

2.7. Análisis de riesgo. Arquitectónico

En la **fase de diseño del sistema se debe de realizar el análisis de riesgos de seguridad con el objetivo de identificar los riesgos del mismo** y clasificarlos en orden de importancia. Después se gestiona y se intenta mitigar el riesgo aplicando las salvaguardas más adecuadas.

El aspecto económico es un factor muy importante a tener en cuenta porque evidentemente las salvaguardas a utilizar para mitigar el riesgo pueden ser en conjunto costosas y por ello es muy crítico priorizar el riesgo y convencer a la alta dirección de su importancia para que dentro de las posibilidades económicas se consiga un sistema lo más libre de riesgos posible, esto algunos autores lo denominan establecimiento de la barra de *bugs* (*setting the bug bar*).

Como una alternativa al acercamiento ad-hoc, Cigital usa el **proceso de análisis de riesgo arquitectónico,** mostrado en la figura 10, que supone un gran conocimiento y desarrolla un acercamiento a un análisis de riesgo implica tres pasos básicos, modelo *de Cigital, Building Security In*:

- **Análisis de resistencia al ataque**. Utiliza patrones de ataque y árboles de ataque para analizar la resistencia al ataque, para ello comprueba una lista de categorías de riesgos según el modelo de análisis de riesgos de Microsoft STRIDE:

    - **Modelado**.
    - **Identificar amenazas**.
    - **Mitigación**.
    - **Validación**.

Figura 9. Metodología para el modelado de amenazas. Extraída de http://www.microsoft.com/security/sdl/adopt/threatmodeling.aspx

- **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 para encontrar nuevos defectos, puntos de conflicto y de inconsistencia**.

- **Análisis de debilidad**. Es un subproceso que ayuda al entendimiento del impacto de dependencias de *software* externo.

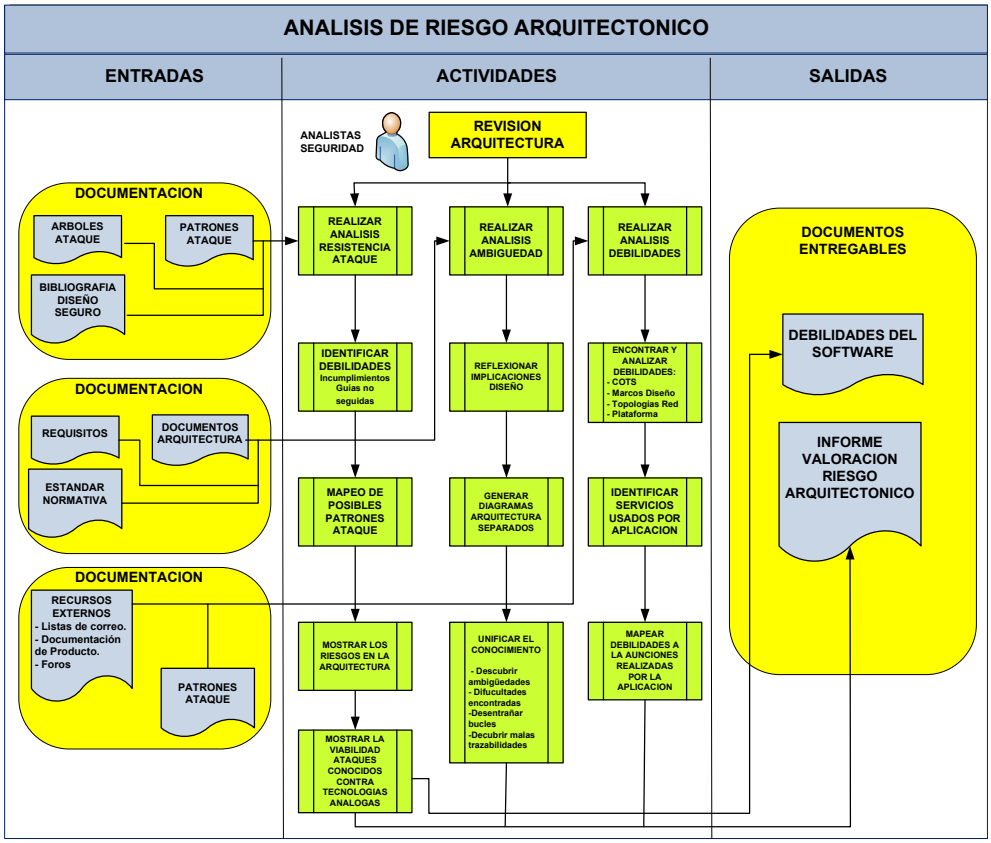


Figura 10. Modelo de Análisis de riesgos arquitectónico de Cigital.
Extraída de McGraw, G. (2005). *Software Security: Building Security In. Addison Wesley Professional*

2.8. Patrones de diseño

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**.

2.9. Pruebas de seguridad basadas en riesgo

Las pruebas de *software* tradicionales no sirven para determinar cómo se comportará en condiciones anómalas y hostiles, ni si está libre de vulnerabilidades.

Identificando los riesgos del sistema y diseñando las pruebas en base a ellos, bajo la perspectiva de un atacante, un **probador de seguridad** de *software* puede enfocar correctamente las áreas de código donde un ataque probablemente pudiera tener éxito.

Las pruebas de seguridad necesariamente deben implicar dos tipos de aproximaciones:

- **Pruebas de seguridad funcionales**: pruebas de los mecanismos de seguridad para asegurar que su funcionalidad está correctamente implementada.
- **Pruebas de seguridad perspectiva atacante**: realización de **pruebas de seguridad en base al riesgo** motivadas por el entendimiento del estado del riesgo y por tanto del nivel de seguridad del sistema. Tratan de simular la actuación de un posible atacante.

Los **objetivos de las pruebas de seguridad basadas en el riesgo** son los siguientes:

- Verificar la operación confiable del *software* bajo condiciones hostiles de ataque.
- Verificar la fiabilidad del *software*, en términos de comportamiento seguro y cambios de estado confiables.
- Verificar la falta de defectos y debilidades explotables.
- Verificar la capacidad de supervivencia del *software* ante la aparición de anomalías, errores y su manejo de las mismas mediante excepciones que minimicen el alcance e impacto de los daños que puedan resultar de los ataques.

Técnicas de test que son particularmente útiles para las pruebas basadas en riesgo incluyen las mostradas en el siguiente diagrama:

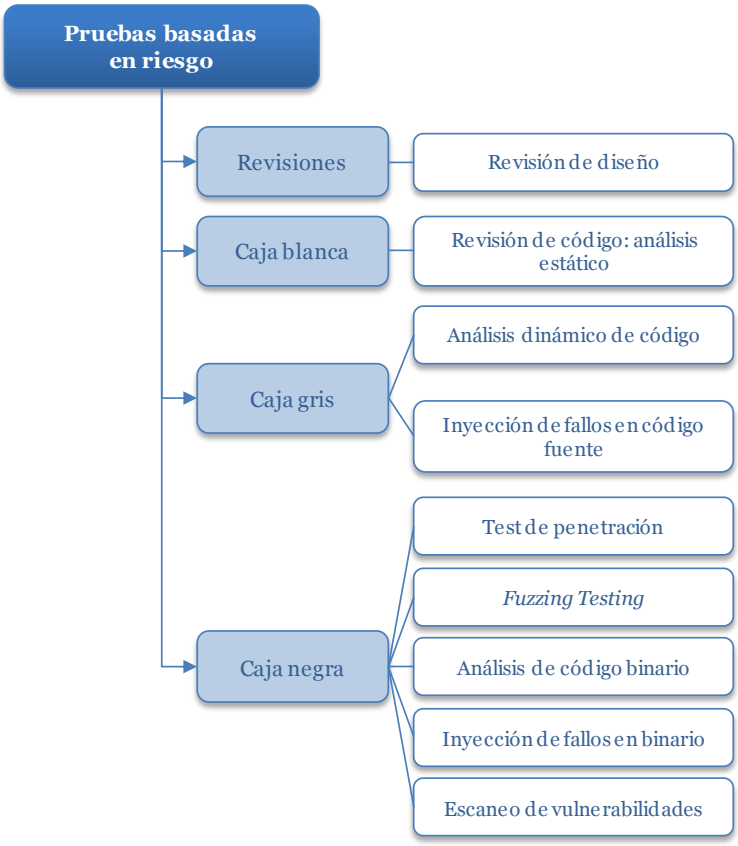


Figura 11. Tipos de pruebas seguridad basadas en riesgo

En la figura siguiente se muestran las diferentes pruebas a realizar en cada una de las fases del S-SDLC:

Figura 12. Distribución pruebas de seguridad a lo largo de las fases del S-SDLC

2.10. Revisión de código

El análisis estático de código fuente se considera **la actividad más importante de entre las mejores prácticas de seguridad** que se han de realizar en el curso del desarrollo de una aplicación.

El **análisis estático de código fuente** es adecuado para identificar problemas de seguridad por ciertas razones:

- Las herramientas de análisis estático comprueban el código a fondo y coherentemente, sin ninguna tendencia.
- A menudo pueden indicar la causa de origen de un problema de seguridad.
- El análisis estático puede encontrar errores tempranamente en el desarrollo, aún antes de que el programa sea ejecutado por primera vez.
- Cuando un investigador de seguridad descubre una nueva variedad de ataque, las herramientas de análisis estático, ayudan a comprobar de nuevo una gran cantidad de código para ver donde el nuevo ataque podría tener éxito.

Los **factores** principales **prácticos** que determinan la utilidad de una herramienta de análisis estático son:

- La capacidad de la herramienta para comprender el programa que se analiza.
- El equilibrio que la herramienta nace entre la precisión, la profundidad y la escalabilidad.
- Porcentaje de falsos positivos y falsos negativos de la herramienta.
- El conjunto de errores que la herramienta comprueba.
- Las características de la herramienta para hacerla fácil de usar.

Figura 13. Diagrama de componentes y proceso de comparación especificación-modelo para una herramienta genérica de análisis de código

El ciclo de revisión de código se muestra en la figura siguiente, la frecuencia con la cual el ciclo es repetido depende en gran parte de los objetivos establecidos. Las cuatro fases principales en el ciclo son:

- **Establecer objetivos**

Figura 14. Ciclo de revisión de código

- **Ejecutar las herramientas de análisis estático**. Para conseguir buenos resultados, se debería compilar el código antes de ser analizado.
- **Revisión del código usando salida de la herramienta**.
- **Hacer correcciones**.

Cuando se usan como parte de la **revisión de código**, las **herramientas de análisis estático pueden ayudar a adoptar las mejores prácticas o *touchpoints***, a **descubrir errores comunes**, y generalmente **hacer el proceso de seguridad más eficiente y consistente**. Para alcanzar estas ventajas, una organización debe tener un proceso de revisión de código bien definido. En un nivel alto, el proceso consiste en cuatro pasos: la definición de objetivos, ejecución de herramientas, revisar el código y hacer correcciones. **Un síntoma de un proceso ineficaz es caer frecuentemente en un debate sobre la explotabilidad**.

En la figura siguiente se resume cómo sería un proceso de análisis estático de código, que enmarcado en el proceso de revisión de código, se divide en tres partes de un sistema informático, con sus documentos de entrada y datos e informes de salida:

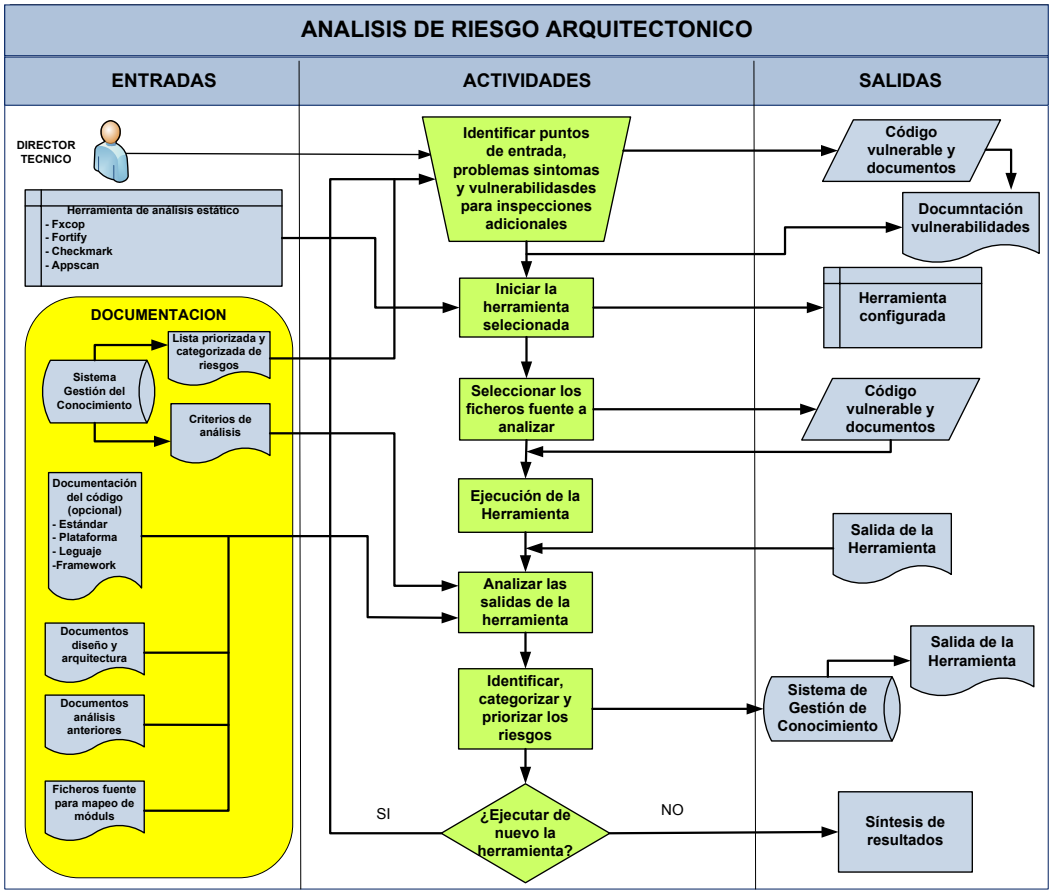


Figura 15. Análisis estático de código, resumen de actividades

- **Entradas**
  - Herramienta de análisis estático.
  - Lista priorizada de riesgos.
  - Código y documentación.
  - Documentos de diseño y arquitectura.
  - Documentación de análisis previos.

- **Actividades**
  - Seleccionar el código fuente a analizar.
  - Instalar y ejecutar la herramienta de análisis.
  - Identificar, clasificar y priorizar los riesgos con un componente de la herramienta de análisis estático.

- **Salidas**
  - Documentación sobre vulnerabilidades.
  - Informes de la herramienta.
  - Lista de riesgos priorizados.

2.11. Test de penetración

La comprobación de la eficacia de las salvaguardas implementadas se realiza principalmente mediante los test de penetración, que tiene como principal misión verificar cómo el *software* se comporta y resiste ante diferentes tipos de ataque.

Las pruebas de penetración están en algún sentido «probando aspectos negativos», es decir debe probar de forma directa y profunda la seguridad de la aplicación en base a los **riesgos de seguridad (conducido por casos de abuso**, **riesgos arquitectónicos y modelos de ataque)**, al objeto de determinar cómo el sistema se comporta ante los ataques.

Los test de penetración son pruebas de caja negra y son comúnmente las más aplicadas de todas las mejores prácticas de seguridad de *software*, son parte del proceso de aceptación de final.

En una fase tardía del ciclo de vida con las pruebas de penetración, los problemas internos del código son destapadas demasiado tarde y las opciones para el remedio tienen restricciones tanto de tiempo, como de presupuesto.

Una gran ventaja de las pruebas de penetración que bien merece mencionarse, es que se posicionan como **pruebas de tipo de caja negra**. Tomando un **sistema en su verdadero ambiente de producción**, los probadores de penetración pueden llegar a descubrir mejor **problemas operacionales y de configuración normalmente pasados por alto durante el desarrollo de *software***. Por ello las pruebas de penetración necesitan ajustarse, no abandonarlas.

Recomendaciones sobre los test de penetración:

- Realizar los test más de una vez.
- Realimentación del resultado de las pruebas de penetración.
- Utilización de Pruebas de Penetración para Evaluar aplicaciones COTS.

2.12. Operaciones de seguridad

El proceso final a llevar a cabo previo al paso a producción de la aplicación segura, son las actividades centrales de:

- Distribución.
- Despliegue.
- Operaciones.

El objetivo de distribución segura es reducir al mínimo las posibilidades de acceso y manipulación del *software* durante la transmisión de un proveedor a su consumidor (envío a través de medios físicos o descarga de red) por los agentes maliciosos.

El *software* puede haber sido diseñado y desarrollado para ser extremadamente seguro, pero no lo será si sus parámetros de configuración no se establecen como el diseñador lo diseñó. De manera similar, los parámetros de configuración de su entorno de ejecución deben ajustarse de modo que el *software* no sea innecesariamente expuesto a amenazas potenciales, a esta tarea se le denomina «bastionado».

Muchos desarrolladores de *software* argumentan que la distribución, despliegue y operaciones no son parte del proceso de desarrollo de *software*. Hay que ajustar los controles de acceso a la red y niveles de sistema operativo, así como diseñar un sistema de registro de eventos, de monitorización, de *backup* y recuperación de los sistemas de ficheros que será lo más eficaz durante operaciones de respuesta ante incidentes. **Los ataques llegarán y estar preparado para defenderse de ellos y para deshacer el daño ocasionado después de que un ataque ha tenido lugar**.

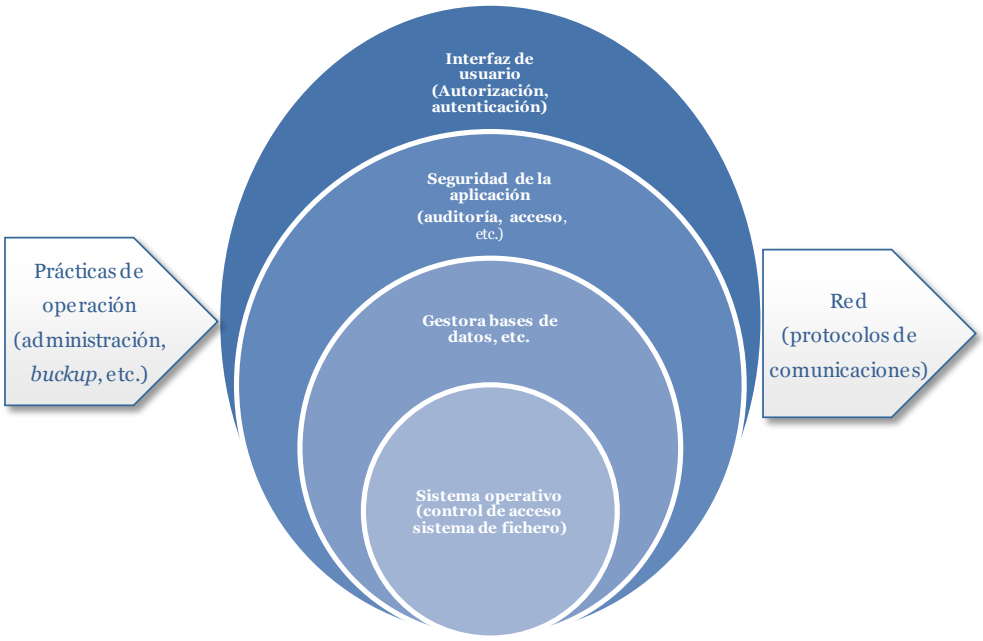


Figura 16. Capas del sistema a proteger

2.13. Revisión externa

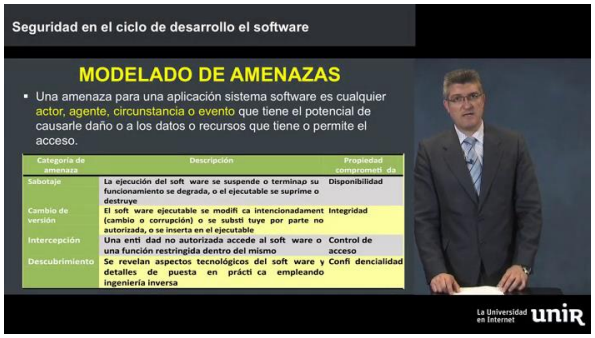
Un análisis externo por personal ajeno al equipo de diseño es bastante eficaz y fundamental aportando otra visión de la seguridad del sistema y del riesgo y contribuyendo a una mejora de la seguridad porque seguramente este análisis va a descubrir alguna amenaza y riesgo residual existente.

Material complementario

Lecciones magistrales

Modelado de amenazas

En la presente clase, se desarrolla la metodología de amenazas propuesta por Microsoft: STRIDE, que puede ser ejecutada tanto por expertos como no expertos en seguridad de la información (diseñadores, desarrolladores, etc.) y se realiza una introducción a la herramienta que la soporta.



El vídeo está disponible en el aula virtual.

No dejes de leer…

Simplified Implementation of the Microsoft SDL

En documento describe los conceptos básicos del *Microsoft Security Development Lifecycle* (SDL) y las actividades de seguridad individuales en cada una de sus fases.

Accede al artículo desde el aula virtual o a través de la siguiente dirección web:
http://www.microsoft.com/en-us/download/details.aspx?id=12379

Building Secure Software

Andrew P. Moore, Robert J. Ellison, Richard C. (2001). *Attack Modeling for Information Security and Survivability*.

Esta nota técnica documenta e identifica patrones de ataque que ocurren comúnmente al objeto de que los diseñadores de sistemas de información y analistas puedan utilizarlos para desarrollar sistemas de información más robustos y confiables.

Accede al artículo desde el aula virtual o a través de la siguiente dirección web:
http://resources.sei.cmu.edu/asset_files/TechnicalNote/2001_004_001_13793.pdf

Software Design Misuse and Abuse Cases

Gary McGraw. *Software Design Misuse and Abuse Cases. Getting Past Positive*. Cigital INC.

Artículo interesante sobre el diseño de casos de abuso.

Accede al artículo desde el aula virtual o a través de la siguiente dirección web:
https://www.cigital.com/papers/download/bsi2-misuse.pdf

No dejes de ver...

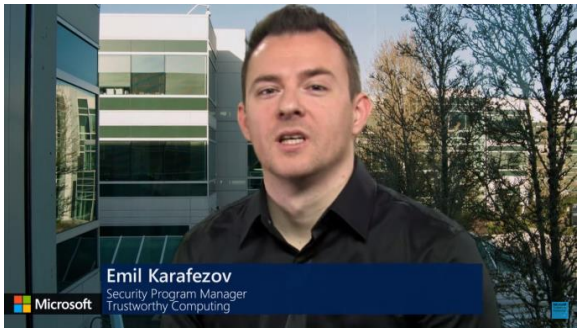
SDL Tools Overview

Vídeo sobre las diversas herramientas a utilizar en las diversas fases de S_SDLC «Microsoft SDL». Doug Cavit, del equipo de Microsoft SDL ingeniería, explica por qué los ejecutivos y gerentes de TI deben alentar a sus equipos de desarrollo para descargar las herramientas para implementar un proceso de aseguramiento de seguridad del *software*, como el SDL Microsoft.

Accede al vídeo desde el aula virtual o a través de la siguiente dirección web:
http://technet.microsoft.com/en-us/video/sdl-tools-overview

Threat Modeling Tool 2014 Demo

Emil Karafezov presenta las nuevas características de la herramienta de modelado de amenazas Microsoft Modeling Tool 2014.



Accede al vídeo desde el aula virtual o a través de la siguiente dirección web:
https://www.youtube.com/watch?v=G2reie1skGg&feature=player_embedded

Threat Modeling

Vídeo que proporciona una descripción del proceso de modelo de amenazas que utiliza Cigital, mediante un ejemplo de aplicación del proceso para identificar fallas potenciales en un sistema.

Objetivos:

- Describir terminología utilizada en el modelado de amenazas.
- Describir un proceso de modelado de amenazas de un sistema.
- Realizar un modelo de amenazas de algún sistema ficticio.

Accede al vídeo desde el aula virtual o a través de la siguiente dirección web:
https://www.youtube.com/watch?v=We2cy8JwVqc

No dejes de practicar…

Aplicación IBM Security AppScan

Demostración de pruebas de caja negra sobre una aplicación web realizada mediante la aplicación IBM Security AppScan. Para ello hay que bajarse la aplicación, instalarla y ejecutarla contra la web de demostración Altoro Mutual website:

Está disponible en la siguiente web, previamente debes rellenar unos formularios:
http://www.ibm.com/developerworks/downloads/r/appscan/

Es una licencia de evaluación que solo puede escanear una página web de prueba, Altoro Mutua en http://demo.testfire.net. Utiliza la plantilla predefinida, demo.testfire.net, que se muestra en el cuadro de diálogo Nueva digitalización. Cuando se te solicite el nombre de usuario y contraseña, utiliza:

- Nombre de usuario: jsmith.
- Contraseña: Demo1234.

La aplicación funciona de forma resumida de la siguiente manera:

Figura 15. Obtenida de Sergio López. Descubriendo el valor de la Seguridad en las Aplicaciones Web con IBM AppScan. IBM Corporation. Marzo 2012.

A fondo

Capturing Security Requirements through Misuse Cases

Guttorm Sindre y Andreas L. Opdahl en este trabajo analizan una extensión conceptual de casos de uso, es decir, casos de mal uso, describen las acciones que no deberían ser posibles en un sistema.

Accede al documento desde el aula virtual o a través de la siguiente dirección web: http://www.nik.no/2001/21-sindre.pdf

Software Assurance in Acquisition: Mitigating Risks to the Enterprise

Esta guía proporciona información sobre cómo incorporar prácticas de aseguramiento del *software* durante todo el proceso de adquisición durante las fases de planificación de la contratación de la adquisición, supervisión y aceptación, y seguimiento. Leer desde la página 12 a la 49.

Accede al artículo desde el aula virtual o a través de la siguiente dirección web:
https://buildsecurityin.us-cert.gov/sites/default/files/SwA_in_Acquisition_102208.pdf

Security Quality Requirements Engineering (SQUARE) Methodology

Nancy R. Mead, Eric D. Hough y Theodore R. Stehney II elaboran este documento para obtener y priorizar los requerimientos de seguridad en los proyectos de desarrollo de *software*. Se detalla y explican los pasos de la metodología y los resultados de su aplicación en estudios de casos recientes.

Accede al artículo desde el aula virtual o a través de la siguiente dirección web:
http://www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA443493

Attack Patterns as a Knowledge Resource for Building Secure

En este trabajo se analiza el concepto de patrones de ataque, como un mecanismo para capturar y comunicar la perspectiva del atacante.

Accede al artículo desde el aula virtual o a través de la siguiente dirección web:
http://capec.mitre.org/documents/Attack_Patterns-Knowing_Your_Enemies_in_Order_to_Defeat_Them-Paper.pdf

Common Attack Pattern Enumeration and Classification (CAPEC)

El propósito de este documento es definir un esquema estándar para representar patrones de ataque y describir de forma detallada el significado y el propósito de cada elemento de esquema constituyente.

Accede al artículo desde el aula virtual o a través de la siguiente dirección web: http://capec.mitre.org/documents/documentation/CAPEC_Schema_Description_v1.3.pdf

Practical Measurement Frameworkfor Software Assurance and Information Security

Documento de métricas de medidas de seguridad del *software* e información que proporciona un método para medir la eficacia de las medidas de seguridad a nivel organizacional programa o proyecto.

Accede al artículo desde el aula virtual o a través de la siguiente dirección web: http://www.psmsc.com/Downloads/TechnologyPapers/SwA%20Measurement%2010-08-08.pdf

Application Security Best Practices at Microsoft: The Microsoft IT Group Shares Its Experiences

El documento contiene la descripción de los procesos, las políticas y mejores prácticas que puedan ser de utilidad para que las organizaciones creen *software* seguro.

Accede al artículo desde el aula virtual o a través de la siguiente dirección web: http://download.microsoft.com/download/0/d/3/0d30736a-a537-480c-bfce-5c884a2fff6c/AppSecurityWhitePaper.doc

Webgrafía

Metasploit

A collaboration of the open source community and Rapid7, Metasploit® software helps security and IT professionals identify security issues, verify vulnerability mitigations, and manage expert-driven security assessments.

Accede a la web desde el aula virtual o a través de la siguiente dirección:
http://www.metasploit.com/

Armitage

Sitio web para descargarse este programa que implementa una onerfaz gráfica de la herramienta de penetración metaexploit. Contiene manuales y vídeos de uso de esta herramienta.

Accede a la web desde el aula virtual o a través de la siguiente dirección:
http://www.fastandeasyhacking.com/

Bibliografía

Allen, J. H.; Barnum, S.; Ellison, R. J.; McGraw, G.; Mead, N. R. (2008). *Software Security Engineering: A Guide for Project Managers*. Addison Wesley Professional.

Chess, B. and West, J. *Secure Programming with Static Analysis*. Addison-Wesley Software Security Series.

Howard, M. and LeBlanc, D. (2003). *Writing Secure Code*. Microsoft Press.

Howard, M. and Lipner, S. (2006). *The Security Development Lifecycle: SDL: A Process for Developing Demonstrably Secure Software*. Microsoft Press.

Krassowski, A. and Meunier, P. (2008). *Secure Software Engineering: Designing, Writing, and Maintaining More Secure Code*. Addison-Wesley.

McGraw, G. (2005). *Software Security: Building Security In. Addison Wesley Professional.*

Redwine, S. T. Jr. (Editor). (2006). S*oftware Assurance: A Guide to the Common Body of Knowledge to Produce, Acquire, and Sustain Secure Software Version 1.1*. US Department of Homeland Security.

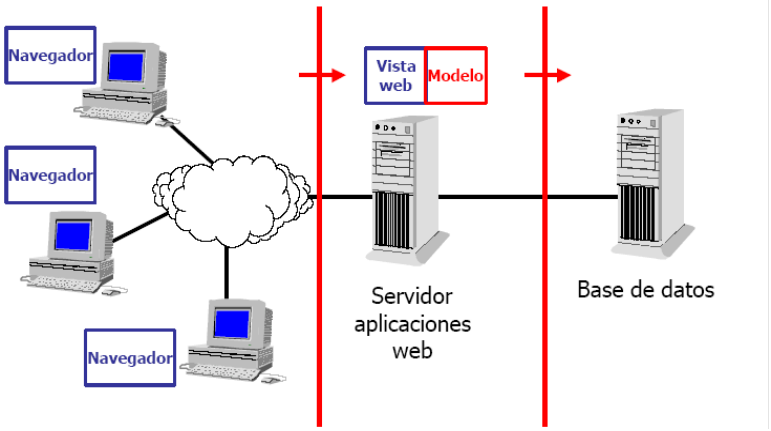
Actividades

Trabajo: Metodologías de modelado de amenazas

El objetivo de esta actividad es seguir profundizando en el modelado de amenazas, realizando un trabajo que contenga al menos el siguiente contenido:

- Introducción al modelado de amenazas.
- Estudio de metodologías existentes.
- Descripción de una metodología en profundidad.
- Ejercicio práctico de modelado de amenazas, utilizando una herramienta de modelado como Threat Analysis and Modeling Tool, del siguiente caso que se describe a continuación (incluir los ficheros generados por la herramienta junto con el del trabajo en un fichero a subir en la plataforma).

Aplicación Web de tres capas para un negocio de pago electrónico de una librería, con la siguiente arquitectura lógica:



Arquitectura de tres capas de una aplicación Web. Extraída de http://oness.sourceforge.net/

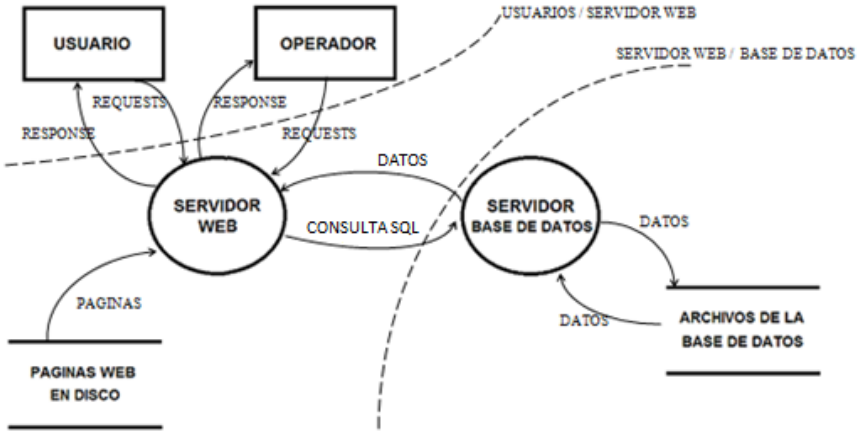
Accede a la herramienta Threat Analysis and Modeling Tool 2016 desde el siguiente enlace:
https://www.microsoft.com/en-us/download/details.aspx?id=49168

Utilizar la aplicación Threat Analysis and Modeling Tool 2016 para aplicar los diferentes pasos de la metodología MTC-SDL descrita en el tema, con el propósito de analizar las amenazas de una aplicación web típica de negocio de pago electrónico de una librería (textos, libros, revistas, etc.), en formato digital con opciones de impresión.

El sistema está basado en una típica arquitectura de una aplicación web de tres capas, donde el cliente es un navegador que acceder a los servicios proporcionados por el sitio web de la librería, que contiene una base de datos de los clientes, cuentas y publicaciones disponibles, alojada en un servidor uno de bases de datos y un servidor web que implementa toda la lógica de negocio.

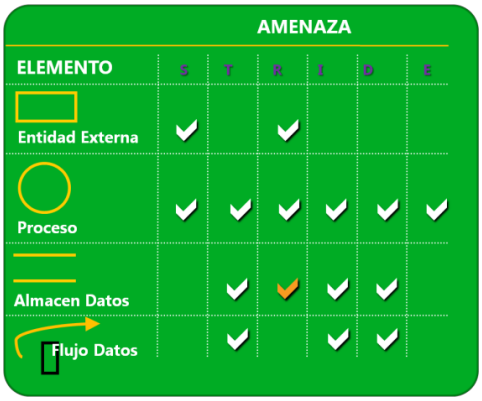
Ten en cuenta que nos encontramos en la fase análisis de requisitos del SDLC, identificando requisitos funcionales y de seguridad.

Os propongo modelar la aplicación mediante el siguiente diagrama DFD que constituye una representación gráfica que agiliza el proceso de modelado de requerimientos y al mismo.



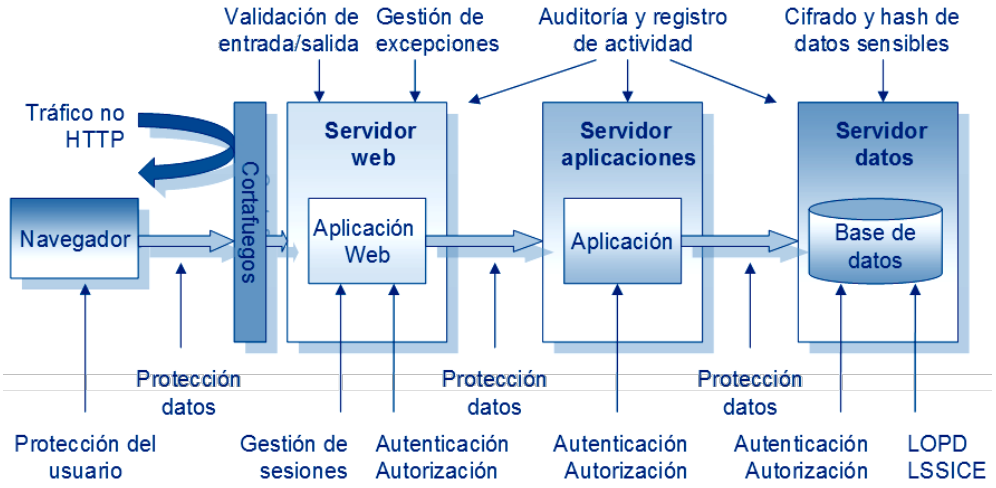
Modelo DFD de la aplicación WEB.

Una vez dibujado el diagrama anterior en la herramienta (u otro que consideres adecuado para tu diseño) produce un análisis automático por cada elemento de la lista de componentes definidos en el diseño, mediante el método STRIDE, tomando como base la siguiente matriz siguiente.



Relación entre las amenazas del método STRIDE y los elementos de un diagrama DFD.

Una vez realizado el análisis automático de las amenazas, hay que incluir manualmente para cada una las salvaguardas que ayuden a mitigarlas. Como ayudas de salvaguardas a incluir en la aplicación, para mitigar las amenazas os incluyo el siguiente dibujo:



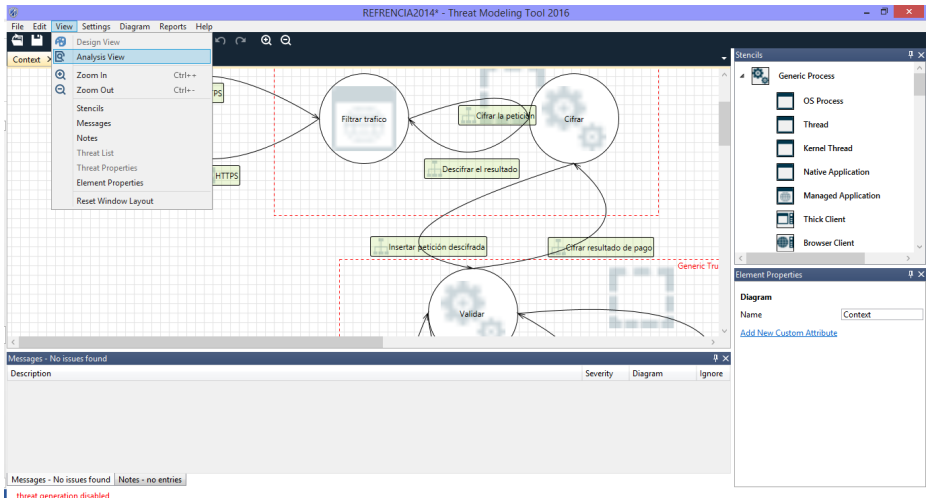
Salvaguardas Aplicación WEB.

Si se quiere un catálogo más completo de salvaguardas consultar el Libro II de la Metodología MAGERIT:

MAGERIT – versión 3.0 Metodología de Análisis y Gestión de Riesgos de los Sistemas de Información Libro II - Catálogo de Elementos. Método. Ministerio de Hacienda y Administraciones Públicas. Disponible: http://administracionelectronica.gob.es/pae_Home/pae_Documentacion/pae_Metodolog/pae_Magerit.html#.U2_oe2CKB2E

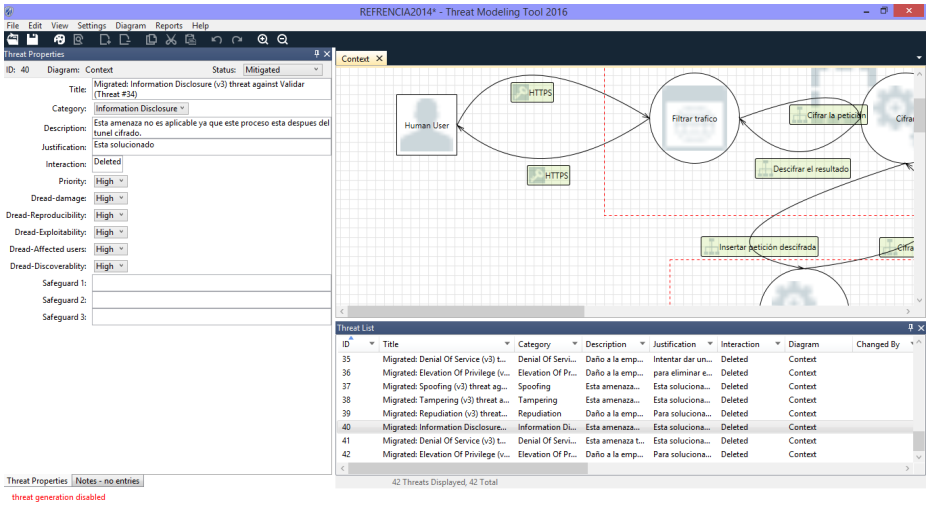
La herramienta Threat Analysis and Modeling Tool 2016 tiene dos vistas:

- Una de diseño para dibujar el diagrama DFD de la aplicación.
- Una de análisis que calcula las amenazas automáticamente y permite incluir las salvaguardas manualmente.



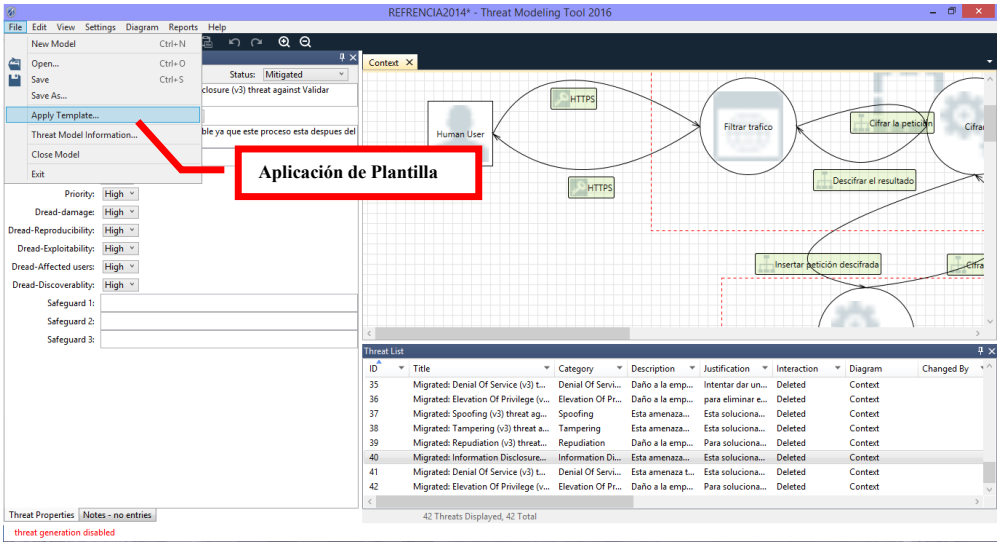
Vistas Threat Analysis and Modeling Tool.

Cuando se selecciona la vista de análisis, la herramienta mostrará las amenazas sugeridas para el diagrama de flujo de datos dibujado, donde al clicar en cada una de ellas nos permite introducir las salvaguardas que consideremos y el análisis DREAD del paso de la metodología.



Vista análisis Threat Analysis and Modeling Tool.

Antes es necesario que cargarla plantilla, descargable de la plataforma, “caso1.tb7” (disponible en el foro de la asignatura) y cargarla mediante el menú «Aply Template», según la figura:



Aplicación de plantilla.

Extensión máxima: 15-20 páginas, fuente Georgia 11 e interlineado 1,5.

Test

1. Señala la o las respuestas correctas. ¿Qué incluye la seguridad del *software*?

A. Patrones de codificación.
B. Principios de diseño seguro.
C. Casos de uso.
D. Buenas prácticas de seguridad.

2. Señala la o las respuestas correctas. Los patrones de ataque en la fase de diseño y arquitectura:

A. Ayudan a definir el comportamiento del sistema para prevenir o reaccionar ante un tipo específico de ataque probable.
B. Especifican debilidades aprovechadas por los ataques, orientando qué técnicas o prácticas de seguridad de desarrollo evitan estas deficiencias.
C. Proporcionan el contexto de las amenazas al que el *software* se va a enfrentar, de forma que permite diseñar arquitecturas seguras.
D. Proporcionan ejemplos de ataques que aprovechan las vulnerabilidades de la arquitectura elegida.

3. Señala la o las correctas. Respecto a los casos de uso de seguridad:

A. Analizan y especifican los requisitos de seguridad.
B. Analizan y especifican las amenazas a la seguridad.
C. Análisis de vulnerabilidades de activos y amenazas.
D. Análisis forense de las actividades maliciosas.

4. Señala la o las respuestas incorrectas. Respecto a la ingeniería de requisitos de seguridad:

A. Requisitos de *software* seguro. Requisitos que afectan directamente a la probabilidad de que el *software* sea seguro.

B. Requisitos de *software* seguro. Especificación de funciones que implementan una política de seguridad, como control de acceso, autenticación, autorización, cifrado y gestión de claves. Estas previenen la violación de la seguridad del sistema o de la información que procesa, como el acceso no autorizado, modificación, denegación de servicio, etc.

C. Los requerimientos no funcionales de seguridad deben especificar: propiedades que el *software* debe exhibir.

D. Los requerimientos no funcionales de seguridad deben especificar: los controles y normas que rigen los procesos de desarrollo, implementación y operación del *software*.

5. Señala la respuesta incorrecta. El análisis de riesgo implica tres pasos básicos:

A. Análisis de ambigüedad.

B. Análisis de resistencia al ataque.

C. Análisis de robustez.

D Análisis de debilidad.

6. Señala la o las respuestas correctas. Las perspectivas de las pruebas de seguridad basadas en el riesgo son las siguientes:

A. Perspectiva gerencia.

B. Perspectiva atacante.

C. Perspectiva usuario.

D Perspectiva defensor.

7. Señala la o las respuestas correctas. El principal problema de las herramientas de análisis estático:

A. Falsos negativos que produce.

B. Gran cantidad de defectos que encuentra.

C. Reglas de la herramienta.

D. Falsos positivos que produce.

8. Señala la o las respuestas correctas. Las herramientas de análisis estático realizan varios tipos de análisis:

A. *Taint Propagation*.
B. Análisis puntual.
C. *Model checking*.
D. Análisis de flujo de datos.

9. Señala la o las respuestas correctas. Los test de penetración:

A. En ningún caso demuestran que ningún defecto existe.
B. Revisan el código.
C. El entendimiento del ambiente de ejecución y de los problemas de configuración es el mejor resultado de cualquier prueba de penetración.
D. Las conclusiones de seguridad no son repetibles a través de equipos diferentes y varían extensamente dependiendo de la habilidad y la experiencia de los probadores.

10. Señala la respuesta correcta. A la hora de realizar la distribución y despliegue del *software* desarrollado, es recomendable el realizar las siguientes buenas prácticas:

A. Distribuir el *software* con una configuración por defecto segura y lo más restrictiva posible.
B. Proporcionar una herramienta de instalación automática.
C. Cambio los valores de configuración predeterminados durante el desarrollo.
D. Todas las anteriores.

