
Diseño de aplicaciones seguras

PID_00208416

Josep Vañó Chic



Los textos e imágenes publicados en esta obra están sujetos –excepto que se indique lo contrario– a una licencia de Reconocimiento-NoComercial-SinObraDerivada (BY-NC-ND) v.3.0 España de Creative Commons. Podéis copiarlos, distribuirlos y transmitirlos públicamente siempre que citéis el autor y la fuente (FUOC. Fundación para la Universitat Oberta de Catalunya), no hagáis de ellos un uso comercial y ni obra derivada. La licencia completa se puede consultar en <http://creativecommons.org/licenses/by-nc-nd/3.0/es/legalcode.es>

Índice

Introducción.....	5
Objetivos.....	6
1. Ciclo de vida del desarrollo de software seguro.....	7
1.1. Formación	8
1.2. Requerimientos	9
1.2.1. Metodologías y estándares	12
1.3. Diseño	13
1.4. Desarrollo	14
1.5. Pruebas	15
1.6. Validación	16
1.7. Mantenimiento	17
1.8. Herramientas	17
2. Evaluación de riesgos.....	20
3. Modelado de amenazas.....	22
3.1. Proceso del modelado de amenazas	23
3.1.1. Identificación de los activos	23
3.1.2. Definición de la arquitectura	23
3.1.3. Descomposición de la aplicación	23
3.1.4. Identificación de las amenazas	24
3.1.5. Mitigación de amenazas	26
3.2. Herramientas	28
4. Técnicas de seguridad.....	31
Bibliografía.....	37

Introducción

Para desarrollar aplicaciones seguras, hay que tener en cuenta la seguridad en todo el proceso de la creación del software, no solo en la fase del desarrollo en la que se escribe código. Se tiene que tener en cuenta la seguridad en todas las fases del proyecto.

Diseñar aplicaciones seguras implica introducir tareas, criterios, documentación, requisitos, análisis, etc. propios de la seguridad en el conjunto de todas las actividades que se llevan a cabo a la hora de crear una aplicación.

En este módulo se hace una aproximación a los aspectos que hay que implementar y tener en cuenta en el diseño de aplicaciones seguras. El enfoque se efectúa desde la perspectiva de la seguridad, es decir, qué elementos se tienen que incorporar en el conjunto de las actividades propias del desarrollo de software para que este sea seguro.

Para diseñar aplicaciones seguras, se lleva a cabo una serie de actividades que están enmarcadas en lo que se denomina *security development lifecycle* (SDL). Hay que tener en cuenta que la aplicación del SDL, en parte, se puede adaptar a las peculiaridades de cada organización, incluso del software en cuestión. En este módulo, el SDL se muestra de una manera genérica para el desarrollo de software en general.

Objetivos

Al finalizar la lectura de este módulo, los estudiantes habrán conseguido las competencias siguientes:

- 1.** Conocer la importancia de la incorporación de la seguridad en el ciclo de vida del software.
- 2.** Conocer las tareas de seguridad en cada fase del ciclo de vida del software.
- 3.** Comprender la importancia del análisis de riesgos.
- 4.** Conocer el proceso del modelado de amenazas.
- 5.** Conocer las técnicas básicas de seguridad.

1. Ciclo de vida del desarrollo de software seguro

El primer paso de una buena práctica de seguridad consiste en ser consciente de que este tema es una parte integral del proceso de desarrollo. Por lo tanto, para que el software cumpla sus objetivos de seguridad, ha de integrarla en el ciclo de vida de desarrollo del software.

La integración de la seguridad puede hacerse mediante la inclusión de determinadas actividades en los procesos actuales de ingeniería de software de la organización, ya sea el modelo en cascada, el modelo iterativo e incremental o el modelo de desarrollo ágil. Esta integración en el ciclo de vida del software se denomina *security development lifecycle*¹

(¹) *SDL: security development lifecycle*.

En este módulo, consideraremos las fases de un ciclo de vida de desarrollo genérico. Hay que tener en cuenta que los procesos de estas fases son casi siempre personalizados para adaptarse a las necesidades del grupo de desarrollo de software² y al modelo de ingeniería elegido.

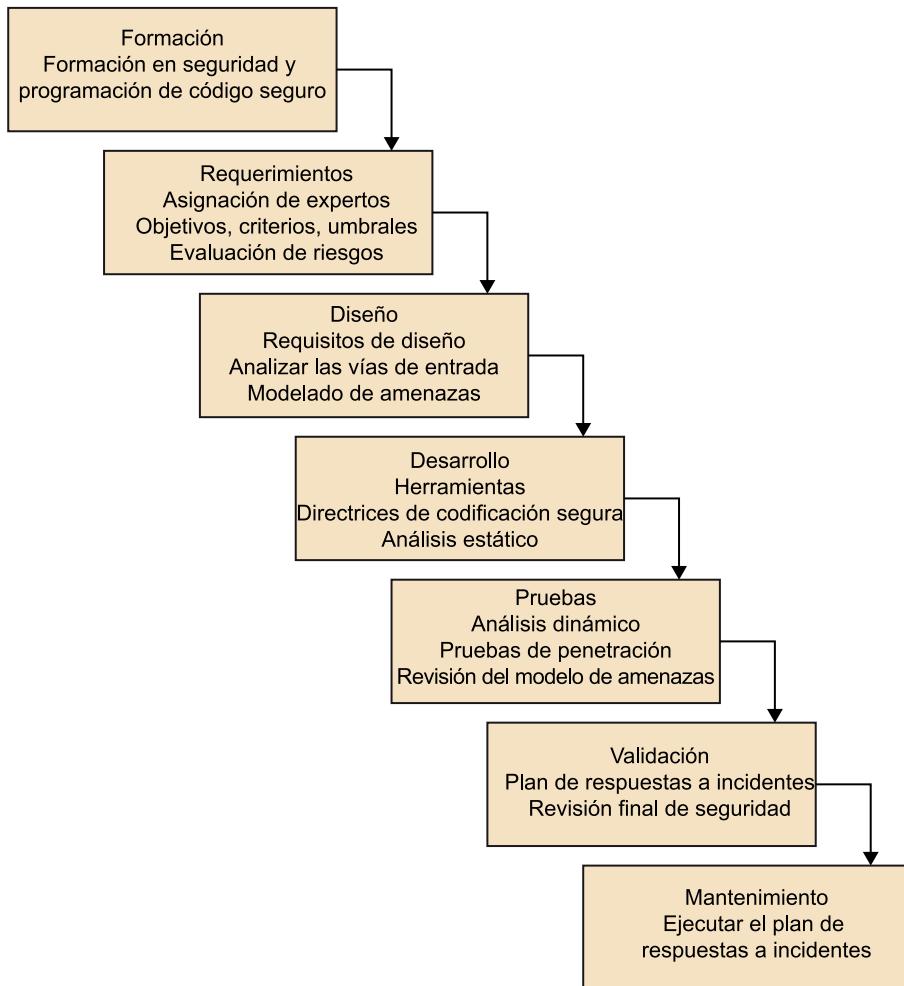
(²) *SDLC: software development lifecycle*.

Fases del ciclo de vida del desarrollo del software:

Bibliografía

Se puede encontrar información variada sobre SDL en Internet.

Podéis consultar The Open Web Application Security Project (OWASP).



1.1. Formación

Comprender los problemas de seguridad es fundamental para desarrollar un software mejor. Con la formación del equipo de desarrollo sobre aspectos básicos y las últimas tendencias en seguridad, así como sobre la programación de código seguro, se consigue que el equipo logre un mejor nivel de concienciación sobre la seguridad, y que aumente el compromiso en la escritura de código seguro. Así pues, resulta importante establecer un plan de formación, que debe ser periódica, para mantener al equipo al día sobre las novedades en seguridad.

Otro aspecto que debemos tener en cuenta es la incorporación de nuevos miembros al equipo. Durante las entrevistas con los miembros potenciales, es necesario hacerles preguntas sobre la seguridad para evaluar su nivel de conocimiento en este aspecto. Esto ayudará a seleccionar a los desarrolladores conscientes en el tema y a evaluar la necesidad de formación adicional sobre la seguridad.

1.2. Requerimientos

Se considera que se necesita diez veces más tiempo, dinero y esfuerzo para corregir un error en la fase de desarrollo que en la fase de diseño, y diez veces más en la fase de pruebas que en la fase de desarrollo. Por lo tanto, el concepto de seguridad debe estar presente desde las primeras fases del ciclo.

Los objetivos y requisitos de seguridad y privacidad se tienen que definir desde el principio del proceso de desarrollo de la aplicación. Si los requisitos se definen en los momentos iniciales, los equipos de desarrollo podrán identificar los principales hitos y resultados e integrar la seguridad y la privacidad, lo que facilitará que la planificación se altere lo menos posible. Los objetivos de la seguridad son propósitos y restricciones que afectan a la confidencialidad, integridad y disponibilidad de los datos y la aplicación. Un software vulnerable pone en peligro estas tres propiedades, que determinan que un sistema informático sea fiable. Así pues, en esta fase se tendrán que contemplar las tareas siguientes:

- Asignación de expertos en seguridad.
- Definición de los objetivos de seguridad.
- Definición de los criterios de seguridad.
- Definición de los umbrales de seguridad y los límites de errores.
- Evaluación de riesgos y análisis de costes.

1) Asignación de expertos en seguridad

Identificar al equipo o a la persona que tendrá la responsabilidad de llevar a cabo el seguimiento y la gestión de la seguridad del producto, así como de coordinar y comunicar el estado de cualquier problema de seguridad.

2) Definición de los objetivos de seguridad

Los objetivos varían en función de varios factores, como por ejemplo, el tipo de usuario, el tipo de datos que se tratarán, el entorno de ejecución, etc.

A pesar de que en una organización se puede establecer un conjunto de categorías de objetivos comunes, hay que considerar que cada aplicación tiene objetivos propios, de modo que es necesario llevar a cabo un proceso de identificación de los objetivos.

Esta identificación se puede iniciar mediante un documento que dé respuesta, por ejemplo, a las preguntas siguientes:

- ¿Cuál es la audiencia de la aplicación?
- ¿Qué tipo de usuario la utilizará?

- ¿Tiene que haber diferentes niveles de requisitos de seguridad en función del usuario?
- ¿Es una aplicación interna de la organización, externa o de los dos tipos?
- ¿Dónde se ejecutará la aplicación? ¿Internet? ¿Red? ¿De qué sistemas de seguridad se dispone? ¿Cortafuegos? ¿DMZ?
- ¿Qué se quiere proteger?
- ¿Qué implicaciones supone el hecho de que se comprometan los objetos que se quiere proteger?
- ¿Qué servicios de infraestructura de seguridad del sistema operativo o del sistema en general se pueden aprovechar?
- ¿Hasta qué punto ha de ser protegido el usuario de sus propios actos?

En el proceso de identificación, los objetivos de seguridad se pueden clasificar en función de las posibles vulnerabilidades de la aplicación y los problemas potenciales, como por ejemplo:

Tipo de objetivo.	Cuestionario.
Valores tangibles que se tienen que proteger:	<ul style="list-style-type: none"> • ¿Hay cuentas de usuario y contraseñas que se deben proteger? • ¿Encontramos información confidencial del usuario (por ejemplo, números de tarjeta de crédito) que necesita ser protegida? • ¿Hay propiedad intelectual sensible que necesita ser protegida? • ¿Puede utilizarse este sistema como un conducto para acceder al resto del sistema de la organización?
Valores intangibles que se tienen que proteger:	<ul style="list-style-type: none"> • ¿Hay valores corporativos que se podrían ver comprometidos por un ataque a este sistema?
Requisitos:	<ul style="list-style-type: none"> • ¿Encontramos políticas de seguridad corporativas que se tienen que cumplir? • ¿Hay legislación sobre seguridad que se tiene que cumplir? • ¿Hay legislación sobre privacidad que se tiene que cumplir? • ¿Encontramos estándares o normas que deben cumplirse? • ¿Hay restricciones o condicionantes debidos al entorno de desarrollo?
Requisitos de la calidad del servicio:	<ul style="list-style-type: none"> • ¿Hay requisitos específicos de disponibilidad que se tienen que cumplir? • ¿Encontramos requisitos específicos de rendimiento que deben cumplirse?

3) Definición de los criterios de seguridad

Se trata de llevar a cabo el análisis de los requisitos de seguridad y privacidad, lo cual debe incluir las especificaciones mínimas de seguridad de la aplicación en el entorno operativo previsto, así como la especificación y la implementación de un sistema de seguimiento de los elementos de trabajo y de las vulnerabilidades de seguridad.

4) Definición de los umbrales de seguridad y los límites de errores

Hay que definir los umbrales de calidad y los límites de errores para establecer los niveles mínimos aceptables de calidad en materia de seguridad y privacidad. Al definir estos criterios al principio del proyecto, se comprenderán mejor los riesgos asociados a los problemas de seguridad y los equipos podrán identificar y corregir los errores de seguridad durante el desarrollo.

5) Evaluación de riesgos y análisis de costes

Implementar sistemas de seguridad y privacidad implica un aumento del coste en el desarrollo, pero no implementar sistemas de seguridad puede suponer un coste mucho más elevado debido, por ejemplo, a la pérdida de información confidencial, a la no disponibilidad del sistema como consecuencia de un ataque que aprovecha una vulnerabilidad u otras posibles situaciones provocadas por un agujero en la seguridad.

Con la evaluación de los riesgos, se identifican los aspectos funcionales del software que requieren una revisión exhaustiva. Aquí se incluye la información siguiente:

- Qué partes del proyecto requerirán modelos de amenaza.
- Qué partes del proyecto requerirán revisiones de diseño de seguridad.
- Qué partes del proyecto requerirán pruebas de penetración por parte de un grupo externo al equipo del proyecto.
- Requisitos de análisis o de pruebas adicionales que se consideren necesarios para mitigar los riesgos de seguridad.

El nivel de privacidad es un aspecto que afecta a la implementación de la seguridad. Por lo tanto, será necesario medir el nivel de impacto sobre ese tema.

- **Alto riesgo de privacidad:** la aplicación, la función, el procedimiento o el servicio almacenan, procesan o transfieren datos de identificación personal, cambian configuraciones o asociaciones de tipos de archivos o instalan software.
- **Riesgo medio:** los datos tratados son anónimos.

- **Riesgo bajo:** no hay nada que afecte a la privacidad ni hay ningún tratamiento de datos personales ni anónimos. No se modifica ninguna configuración ni se instala ningún software.

1.2.1. Metodologías y estándares

La complejidad de llevar a cabo un plan de seguridad y, en consecuencia, tenerla en cuenta desde el principio del ciclo de vida del software hace que sea necesario el uso de una metodología.

Existen varios sistemas de gestión de seguridad de la información (SGSI), como por ejemplo MAGERIT y el estándar ISO 27002³ (antes ISO 17799), entre otros.

⁽³⁾ISO 27002 forma parte de la serie de normas ISO 27000, que proporcionan un marco de gestión de la seguridad de la información.

La ISO 27002 establece un conjunto de dominios de control que cubren completamente la gestión de la seguridad de la información, y en el que cada dominio se refiere a un aspecto de la seguridad de la organización. Concretamente, en el caso del desarrollo de software seguro, se considerarán los aspectos del apartado 12, “Adquisición, desarrollo y mantenimiento de los sistemas de información”, que se desglosa en los puntos siguientes.

12. Adquisición, desarrollo y mantenimiento de los sistemas de información

12.1 Requerimientos de seguridad de los sistemas de información

12.1.1 Análisis y especificación de los requisitos de seguridad

12.2 Procesamiento correcto de las aplicaciones

12.2.1 Validación de los datos de entrada

12.2.2 Control del procesamiento interno

12.2.3 Integridad de los mensajes

12.2.4 Validación de los datos de salida

12.3 Controles criptográficos

12.4 Seguridad de los archivos del sistema

12.4 Seguridad en los procesos de desarrollo y apoyo

12.6 Gestión de la vulnerabilidad técnica

MAGERIT⁴ es una metodología pública⁵ desarrollada por el Ministerio de Administraciones Públicas y cubre los aspectos del análisis y la gestión de riesgos de los sistemas de información. Al mismo tiempo se basa en varias normas, por ejemplo, entre otras, la ISO 27002. Esta metodología consta de cuatro fases:

- Planificación del análisis y la gestión de riesgos.
- Análisis de riesgos.
- Gestión del riesgo.

⁽⁴⁾MAGERIT es el acrónimo de metodología de análisis y gestión de riesgos de los sistemas de información.

⁽⁵⁾Su documentación está disponible en el portal del Ministerio de Administraciones Públicas.

- Selección de salvaguardas.

1.3. Diseño

En la fase de diseño, se incluyen las especificaciones funcionales y de diseño en términos de seguridad. Además, también se lleva a cabo el análisis de riesgos y el modelado de amenazas para identificar las amenazas y vulnerabilidades en el software.

En las especificaciones funcionales se describen las características de seguridad o funciones de privacidad que están directamente expuestas a los usuarios, como por ejemplo, la necesidad de autenticación del usuario para acceder a datos específicos o el consentimiento del usuario antes de ejecutar una función de privacidad de alto riesgo. En las especificaciones de diseño, se describe cómo implementar estas características y funcionalidades y, además, cómo hacerlo de manera segura.

En esta fase, se contemplan las tareas siguientes:

- Establecer los requisitos del diseño de la seguridad.
- Analizar las vías de entrada de los atacantes.
- Modelado de amenazas.

1) Establecer los requisitos del diseño de la seguridad

Algunos ejemplos de las actividades en este punto son:

- Creación de las especificaciones de diseño en materia de la seguridad y privacidad.
- Especificaciones de los requisitos mínimos del diseño criptográfico.
- Descripción de las características de seguridad o privacidad que serán expuestas a los usuarios.
- Descripción de cómo implementar de manera segura las funcionalidades y las distintas características del software.

2) Analizar las vías de entrada de los atacantes

Los ataques aprovechan los puntos débiles y las vulnerabilidades. Por lo tanto, es importante limitar las posibilidades de acceso a los atacantes y, de este modo, reducir los riesgos. En este caso, hay que tener en cuenta, por ejemplo, los aspectos siguientes:

- Cerrar o restringir el acceso a los servicios del sistema.
- Aplicar el principio de privilegios mínimos.

- Configurar correctamente las excepciones del cortafuegos.

3) Modelado de amenazas

Este procedimiento permite a los equipos de desarrollo considerar, documentar y describir de manera estructurada las implicaciones del diseño en la seguridad. La creación del modelado de amenazas es un trabajo en equipo que implica a los diseñadores, desarrolladores, el equipo de pruebas, los técnicos en seguridad, etc. Esta es la tarea principal de análisis de seguridad que se lleva a cabo en la fase de diseño de software, a la que se destina un apartado en este módulo.

1.4. Desarrollo

El desarrollo implica escribir y depurar código, y el objetivo es escribir código con la mejor calidad posible. La calidad está relacionada con la seguridad: no se puede decir que un código es de calidad si no incorpora la seguridad. En este apartado se tendrán que considerar los puntos siguientes:

- Herramientas de desarrollo.
- Directrices de codificación segura.
- Análisis estático.

1) Herramientas de desarrollo

El equipo de desarrollo tiene que definir una lista de herramientas aprobadas y de las comprobaciones de seguridad asociadas, como por ejemplo las advertencias y las opciones del compilador. Esta lista la tiene que aprobar el asesor o responsable de seguridad del proyecto. En general, se tiene que procurar utilizar la última versión de las herramientas aprobadas para aprovechar las nuevas protecciones y funciones de análisis de seguridad.

2) Directrices de codificación segura

Es necesario definir un conjunto mínimo de normas de codificación, analizar las bibliotecas, funciones y las API que se pueden utilizar de manera conjunta en un proyecto de desarrollo de software y, además, prohibir las que se determine que no son seguras.

3) Análisis estático

Además de la revisión de código efectuada manualmente, hay que tener en cuenta el análisis estático, que proporciona una capacidad escalable para la revisión de código seguro y puede ayudar a comprobar que se sigan las políticas de codificación segura. Aun así, debemos ser conscientes de las fortalezas y

debilidades de las herramientas de análisis estático y de que es necesario añadir una revisión humana. Por lo tanto, es preciso considerar las herramientas de análisis estático como herramientas de ayuda y no como sustitución de otros métodos de análisis y revisión.

1.5. Pruebas

La función de las pruebas de seguridad es verificar que los diseños del sistema y del código pueden resistir el ataque. Por lo tanto, en esta fase se llevarán a cabo las tareas siguientes:

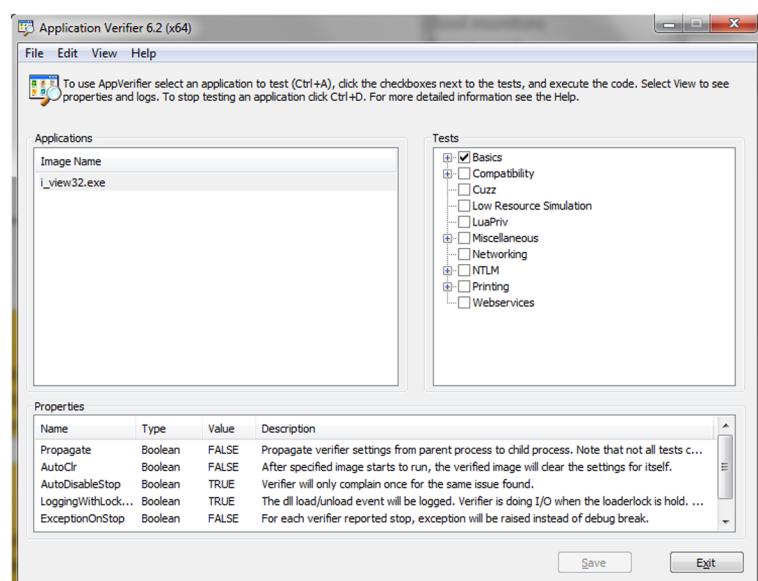
- Análisis dinámico.
- Pruebas de penetración.
- Revisión del modelo de amenazas.

1) Análisis dinámico

Es necesario comprobar el software desarrollado en tiempo de ejecución para asegurar que su funcionalidad se corresponde con el diseño.

En esta etapa se tiene que considerar el uso de herramientas, como por ejemplo AppVerifier⁽⁶⁾, que supervisen el comportamiento de las aplicaciones, y que de este modo sea posible detectar las posibles corrupciones de memoria, problemas con los privilegios de los usuarios u otro tipo de problemas de vulnerabilidad críticos. También hay que tener en cuenta el uso de otros métodos, como por ejemplo las pruebas de exploración de vulnerabilidades con datos aleatorios, datos con formato incorrecto, datos fuera del rango previsto, etc.

⁽⁶⁾Es una herramienta de Microsoft diseñada de manera específica para detectar y ayudar a depurar corrupciones en la memoria y vulnerabilidades críticas en la seguridad.



2) Pruebas de penetración

Son de vital importancia cuando el software se utiliza en escenarios críticos. Se trata de pruebas de caja blanca que simulan las acciones de un *hacker*. El objetivo de estas pruebas es detectar vulnerabilidades debidas a errores de codificación, configuración u otros puntos débiles en la implementación.

3) Revisión del modelo de amenazas

Con frecuencia, una aplicación se desvía de manera significativa de las especificaciones funcionales y de diseño definidas en las fases de requisitos y de diseño. Por lo tanto, es importante llevar a cabo una revisión de lo que se elaboró en la fase de requisitos y de diseño para actualizar el modelo de amenazas según los cambios que se hayan producido. De este modo, se podrán tener en consideración las nuevas posibles amenazas y vulnerabilidades que pueden aparecer con los cambios de diseño y revisarlas y mitigarlas con los criterios actualizados.

1.6. Validación

La implementación requiere una serie de tareas para poner finalmente el software en producción, entre las cuales se tendrán en cuenta las siguientes:

- Llevar a cabo el plan de respuestas a incidentes.
- Revisión final de seguridad.

1) Plan de respuestas a incidentes

A pesar de que se hayan superado todas las pruebas, una vez se ha puesto la aplicación en producción es posible que aparezcan nuevas amenazas hasta ahora desconocidas, o que no se hayan previsto o detectado algunos tipos de amenazas. De este modo, no se tiene que descartar que una vez el software esté en producción se detecten vulnerabilidades que se tengan que corregir, por lo que es preciso elaborar un plan de respuestas a incidentes. Este plan varía en función del tipo de organización y del tipo de software, pero a modo de ejemplo podemos indicar los siguientes:

- Identificar un equipo de mantenimiento y uno de emergencias, si es necesario, que den respuesta a las vulnerabilidades.
- Plan de servicios de seguridad para código heredado de otros equipos de la organización.
- Si se da el caso, plan de seguridad para código de terceros bajo licencia.

2) Revisión final de seguridad

Se trata de inspeccionar las actividades de seguridad efectuadas antes de su lanzamiento. En este punto no se trata de volver a hacer un test o de corregir errores, sino de evaluar los resultados de las herramientas, las pruebas y el rendimiento, teniendo en cuenta los umbrales de calidad y límites de errores previamente determinados. La revisión de seguridad puede tener tres resultados:

- **Revisión superada:** se han corregido y mitigado todos los problemas de seguridad y privacidad identificados.
- **Revisión superada con excepciones:** se han corregido y mitigado los problemas de seguridad y privacidad identificados. Aun así, se detectan problemas que no se han podido resolver, que se registran y se prevé corregir en una versión posterior.
- **Revisión con remisión a una instancia superior:** no se cumplen o no se han corregido los problemas de seguridad y privacidad con unos mínimos aceptables. El responsable de seguridad no puede aprobar el proyecto y no es posible ponerlo en producción. Por lo tanto, se tiene que remitir a una instancia superior para la consiguiente toma de decisiones.

1.7. Mantenimiento

A lo largo de la vida del software, se van efectuando modificaciones, ya sean correctivas, adaptativas, evolutivas o perfectivas. Estas modificaciones tienen que ser gestionadas con lo que se denomina **gestión del cambio**.

En esta fase de mantenimiento, no nos tenemos que olvidar de la seguridad y hay que seguir incorporando todos los elementos de seguridad y privacidad que se han tratado hasta ahora. En la gestión del cambio se tienen que incorporar todas las tareas y los elementos necesarios para que las modificaciones cumplan con los requisitos de seguridad y, además, que la aplicación continúe cumpliendo los umbrales de seguridad y límites de errores estipulados en el proyecto.

1.8. Herramientas

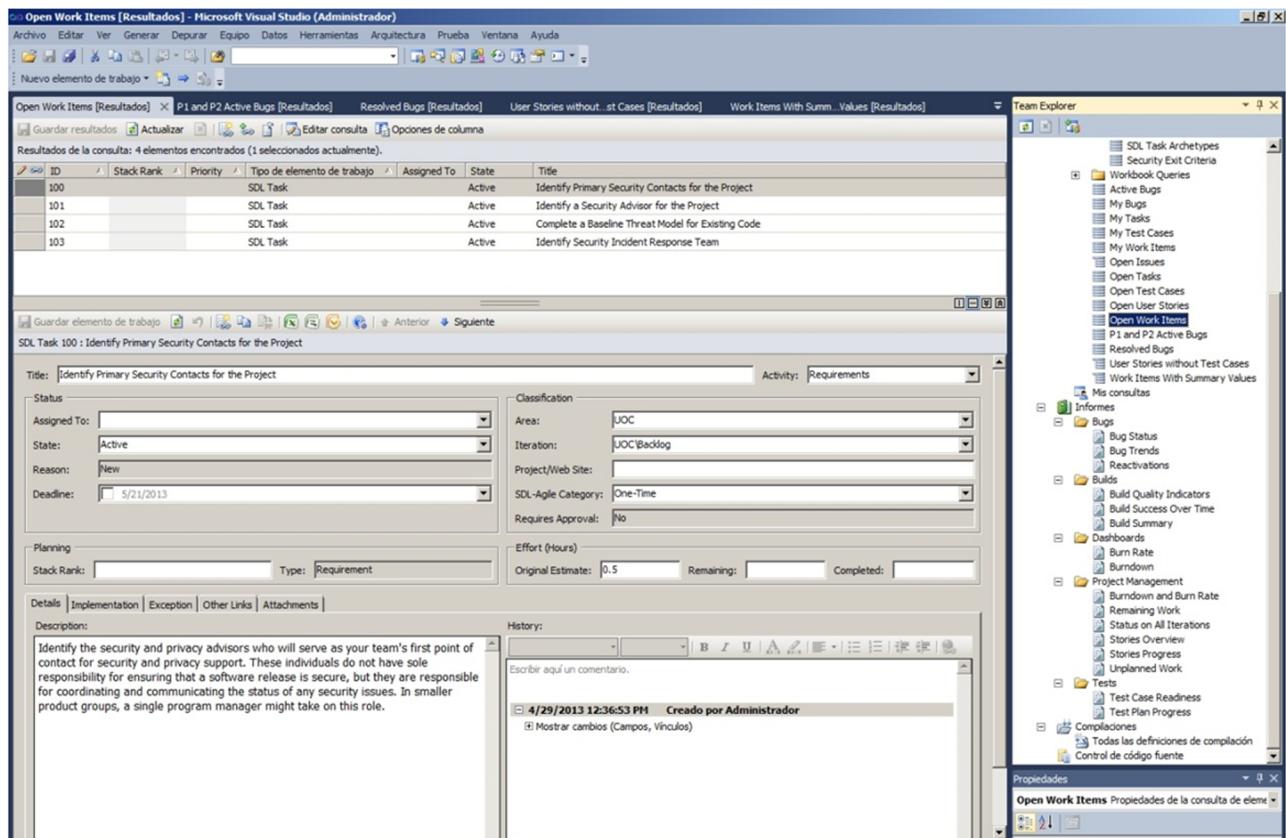
Incorporar la seguridad en el ciclo de vida del software implica gestionar, documentar, controlar y hacer el seguimiento de todo un conjunto de tareas durante todo el proceso. En el mercado hay varias herramientas para las distintas plataformas de desarrollo, que integran la seguridad en todas sus fases, desde la de requisitos hasta la de mantenimiento. Como ejemplo, exponemos la herramienta de Microsoft MSF-Agile plus SDL Process Template for VSTS, que está disponible para diferentes versiones de Visual Studio .NET con Team Foundation Server. Se trata de unas plantillas para el modelo de metodologías

Bibliografía

Las plantillas MSF-Agile Plus SDL Process Template for VSTS se pueden obtener en la web de Microsoft.

Podéis ampliar información sobre estas plantillas en la misma web de Microsoft.

ágiles y para el modelo iterativo e incremental. Con estas plantillas, se analiza el código desarrollado para asegurar que cumple con las prácticas de desarrollo seguro. Las plantillas también crean de manera automática el flujo de trabajo de seguridad de los elementos de seguimiento de procesos, como el del modelado de amenazas, para asegurar que estas actividades de seguridad importantes no sean omitidas.



SDL Task Archetypes [Resultados] - Microsoft Visual Studio (Administrador)

Archivo Editar Ver Generar Depurar Equipo Datos Herramientas Arquitectura Prueba Ventana Ayuda

Nuevo elemento de trabajo Guardar resultados Actualizar Editar consulta Opciones de columna

Guardado resultados Resolved Security Bugs [Resultados]

Resultados de la consulta: 37 elementos encontrados (1 seleccionados actualmente).

ID	Title	Task Type
104	Ensure that the Team has Completed Training	Requirement
106	Strongly Name Assemblies and Request Minimal Permissions	Requirement
107	Review Use of Cryptography	Requirement
111	Disable Tracing and Debugging for ASP.NET Applications Before Deployment	Requirement
114	Complete Threat Models for New Features	Requirement
115	Enable /GS	Requirement
116	Enable Address Space Layout Randomization (ASLR)	Requirement
117	Enable Data Execution Prevention (DEP)	Requirement
118	Enable Safe Structured Exception Handling	Requirement
119	Enable /ROBUST MIDL Compilation	Requirement
120	Enable Visual Studio Code Analysis	Requirement
121	Enable Visual Studio Code Analysis	Requirement
122	Fix any Issues Identified by CAT.NET	Requirement

SDL Task Archetype 104 : Ensure that the Team has Completed Training

Title: Ensure that the Team has Completed Training Activity:

Applicability

Type:	Requirement	Frequency:	PerIteration
Native:	Yes	Web:	Yes
Managed:	Yes	Windows:	Yes

Effort (hours)

Original Estimate:

Exception Rating: 3 - Medium

Requires Security Reviewer Approval: No

Description:

All developers, testers, and program managers must complete at least one security training class each year. Individuals who have not taken a class in the basics of security design, development, and testing must do so. At least 80 percent of the project team staff who work on the product must be in compliance with this standard. Relevant managers must also be in compliance.

Team Explorer

win2008|DefaultCollection

- Mis favoritos
- UOC
- Elementos de trabajo
 - Consultas del equipo
 - Security Queries
 - Active Security Bugs
 - Approved Exceptions
 - My SDL Tasks
 - My Security Bugs
 - Open Exceptions
 - Open SDL Tasks
 - Resolved Security Bugs
 - SDL Task Archetypes
 - Security Exit Criteria
 - Workbook Queries
 - Active Bugs
 - My Bugs
 - My Tasks
 - My Test Cases
 - My Work Items
 - Open Issues

2. Evaluación de riesgos

Se tiene que considerar que el riesgo, poco o mucho, siempre estará presente. Esto hace que en los proyectos se deba gestionar el riesgo y llevar a cabo un análisis de riesgos. Dado que el riesgo existe, se trata de establecer mecanismos de observación y prevención, planes para mitigar o reducir los riesgos y planes de contingencias para el caso en que este deje de serlo, se active y se transforme en un problema real.

En el desarrollo de software, como en cualquier proyecto, las fases del análisis de riesgos son las mismas que en otros ámbitos. Las metodologías que se pueden aplicar son diferentes, como por ejemplo MAGERIT, NIST, CRAMM, OCTAVE, etc. En general, las fases del análisis se podrían resumir de esta manera:

- Definición y valoración de los activos afectados.
- Identificación de las vulnerabilidades.
- Identificación de las amenazas.
- Estudio de las salvaguardas.
- Determinación de la probabilidad.
- Análisis de impacto.
- Determinación del nivel de riesgo.

A continuación se exponen brevemente algunas consideraciones muy generales que se deberían tener en cuenta en el análisis de riesgos en el caso del desarrollo de software.

En primer lugar, para llevar a cabo una evaluación de riesgos se tiene que asumir que el software o el sistema serán atacados.

La cantidad de tiempo y esfuerzo que destinará un sujeto a intentar encontrar cómo atacar un sistema depende de varios factores, entre los cuales se pueden plantear los siguientes:

- El valor de los datos que trate el software, como por ejemplo, números de tarjetas de crédito, datos personales que pueden ser utilizados, datos económicos, etc.
- ¿Es el software de uso de una pequeña empresa o de una multinacional?
- ¿Cuál será la distribución del software, de uso exclusivo para un cliente o se trata de una aplicación estándar de uso generalizado?

- ¿Cuál será el entorno de ejecución, en una red local o de acceso por medio de Internet?

Teniendo en cuenta estos factores, además de los que se determinen, es preciso decidir qué nivel de riesgo resulta aceptable. Una posible pérdida de 1.000 euros no justifica una inversión añadida en desarrollo de 10.000 euros para prever todos los errores potenciales de seguridad. Pese a tener en cuenta este criterio, también hay que considerar el coste en concepto de pérdida de reputación que puede sufrir la empresa, con la consiguiente pérdida de clientes actuales y potenciales.

Evaluar el riesgo

La evaluación de riesgos depende mucho del tipo de software. Aun así, algunos de los factores que se pueden considerar son los siguientes:

- ¿Qué es lo peor que puede pasar si la aplicación es atacada con éxito?
- ¿Qué grado de dificultad tendría el atacante para conseguir un ataque con éxito?
- ¿Cuál sería el tamaño del objetivo del atacante? ¿Se trata de una aplicación con centenares de copias vendidas o está instalada por defecto en miles de ordenadores?
- ¿Es vulnerable por defecto, lo es cuando el usuario lleva a cabo un conjunto inusual de opciones o se necesitan herramientas especializadas?
- ¿Cuántos usuarios se verían afectados?
- ¿Cómo se accede al objetivo? ¿Requiere la ejecución acceso a la red local, acceso local o acceso por medio de Internet?

Una evaluación de riesgos da una idea de la probabilidad de ser atacados y del daño que podría causar un ataque. El siguiente paso consiste en averiguar cómo se puede ser atacado, incluido el sistema: no solo los atentados al software, sino también a los servidores, datos, etc. Para efectuar esta tarea, se crea un modelo de amenazas.

3. Modelado de amenazas

El modelado de amenazas es una técnica que se puede utilizar para ayudar a identificar amenazas, ataques, vulnerabilidades y contramedidas relevantes para la aplicación, de modo que permite determinar mejor los riesgos a los que puede ser expuesto el software y cómo se pueden manifestar los ataques. El objetivo es determinar qué amenazas requieren ser mitigadas y cómo hacerlo. La actividad de modelado de amenazas ayuda a identificar lo siguiente:

- Los objetivos de seguridad.
- Las amenazas relevantes.
- Las vulnerabilidades relevantes y sus contramedidas.

Básicamente, el modelado de amenazas consiste en revisar el diseño y/o la arquitectura siguiendo una metodología que facilite encontrar y corregir los problemas de seguridad actuales y/o futuros. Se trata de que los diferentes actores involucrados (desarrolladores, equipo de pruebas, gerencia, administradores de sistemas, etc.) participen en el proceso de identificación de las posibles amenazas, tanto tomando una actitud defensiva como poniéndose en el papel de un posible atacante. Siguiendo este enfoque, se fuerza a todos los mismos a explorar las debilidades que puedan surgir o que ya estén presentes, así como a determinar las posibles contramedidas.

Al mismo tiempo, la elaboración de un modelado de amenazas permite identificar los objetivos de seguridad específicos de cada entorno y cumplirlos, y facilita la priorización de tareas según el nivel de riesgo.

El modelado de amenazas es un proceso iterativo. Hay que tener en cuenta que es imposible identificar todas las posibles amenazas en una sola pasada y, además, las aplicaciones rara vez son estáticas, sino que se van mejorando y adaptando a las nuevas necesidades de la organización. Por lo tanto, a medida que el ciclo de vida avanza, de manera progresiva se va añadiendo más detalle al modelo:

- Incrementar el detalle del modelo a medida que se descubren nuevos factores.
- Mientras se desarrolla, las decisiones de diseño e implementación revelan nuevos factores.
- Hay que tener en cuenta que, a través de las distintas fases del ciclo de vida del desarrollo del software, pueden aparecer nuevos factores, incluso

en la fase de mantenimiento, así como al configurar el sistema y con el posterior uso de la aplicación.

3.1. Proceso del modelado de amenazas

El proceso del modelado de amenazas se compone de las fases siguientes:

- Identificación de los activos.
- Definición de la arquitectura.
- Descomposición de la aplicación.
- Identificación de las amenazas:
 - Identificación.
 - Documentación.
 - Valoración.
- Mitigación de las amenazas.

3.1.1. Identificación de los activos

Identificar los activos que se tienen que proteger. Estos pueden ser datos confidenciales, bases de datos, páginas web, disponibilidad del sitio web o del sistema, etc.

3.1.2. Definición de la arquitectura

El objetivo de esta etapa es documentar el funcionamiento de la aplicación, su arquitectura y configuración de implementación, así como las tecnologías que forman parte de la solución. Todo esto debe hacerse buscando posibles vulnerabilidades en el diseño o en la implementación de la aplicación.

3.1.3. Descomposición de la aplicación

El objetivo de esta fase es conseguir una comprensión de la aplicación y de cómo interactúa con las entidades externas, para de este modo descubrir las amenazas con más facilidad.

En esta etapa se llevan a cabo las tareas siguientes:

- Identificar los límites de confianza.
- Analizar y efectuar un diagrama de flujo de datos.
- Identificar los puntos de entrada.
- Identificar código privilegiado.
- Documentar el perfil de seguridad.

3.1.4. Identificación de las amenazas

En este punto se identifican las amenazas que pueden afectar al sistema y comprometer los activos. Una de las maneras habituales de llevar a cabo esta tarea es reunir a un grupo con miembros de los equipos de desarrollo y de pruebas, y efectuar una sesión de lluvia de ideas.

1) Identificar las amenazas

STRIDE⁷ es un sistema desarrollado por Microsoft para la clasificación de las amenazas de seguridad.

⁽⁷⁾STRIDE es el acrónimo de *spoofing, tampering, repudiation, information disclosure, denial of service, elevation of privilege*.

Para identificar las amenazas, podemos utilizar tres enfoques básicos:

a) Utilizar STRIDE para categorizar las amenazas.

En el proceso de identificación de amenazas, es importante hacerse preguntas como las siguientes:

- ¿Podría un usuario no autorizado visualizar datos confidenciales?
- ¿Podría un usuario con privilegios solo de lectura modificar registros en la base de datos?
- ¿Podría un usuario utilizar algún componente para elevar sus privilegios a los de administrador?

Para facilitar la formulación de este tipo de preguntas, podemos utilizar el sistema STRIDE y agruparlas en seis categorías.

- ***Spoofing***: esta amenaza permite a un atacante hacerse pasar por otro, ya sea un usuario o un servidor.
- ***Tampering***: la manipulación de datos implica la modificación maliciosa de los mismos.
- ***Repudiation***: esta amenaza implica que un usuario puede negar que ha llevado a cabo una acción que en realidad ha hecho, y esto no se puede demostrar, es decir, repudiar.
- ***Information disclosure***: divulgación de información a personas no autorizadas.
- ***Denial of service***: denegación de servicio o acceso a usuarios válidos.

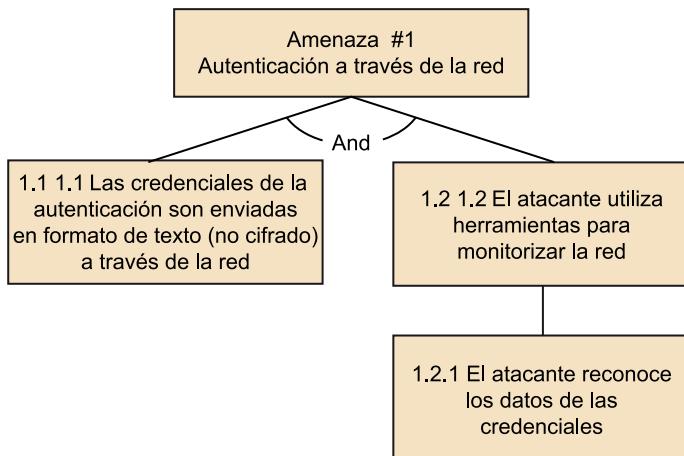
- ***Elevation of privilege***: en este caso, un usuario puede elevar sus privilegios de usuario y, de este modo, llevar a cabo tareas para las cuales no estaba autorizado.

b) Utilizar listas de amenazas clasificadas.

Con este enfoque, se crea en primer lugar una lista de amenazas comunes agrupadas por tipos (por ejemplo, por red, servidor y la aplicación). A continuación, se aplica la lista de amenazas a la propia arquitectura de la aplicación.

c) Árboles de amenazas.

Con este sistema, se trata de descomponer una amenaza con las distintas formas que un atacante puede utilizar.



2) Documentar las amenazas

El siguiente paso en la identificación consiste en documentar las amenazas. Para hacerlo, se utilizará una plantilla en la que se incluyen los apartados siguientes:

- Descripción de la amenaza.
- Objetivo de la amenaza.
- Nivel de riesgo.
- Técnicas de ataque.
- Contramedidas.

3) Valoración de las amenazas

Una vez identificadas y documentadas las amenazas, es necesario determinar las más importantes para priorizar los esfuerzos que hay que destinar a las mismas. Para clasificarlas en función de su importancia, se lleva a cabo una valoración de cada una, de modo que se pueda cuantificar. En cuanto se hayan valorado, se clasificarán en orden decreciente.

Una forma sencilla de hacer la valoración consiste en puntuar la probabilidad y su daño potencial, con valores del 0 al 10 y aplicar la fórmula siguiente:

$$\text{Riesgo} = \text{Probabilidad} * \text{Daño potencial}$$

En este caso, se obtiene una valoración del riesgo con un intervalo del 0 al 100.

Otra manera de llevar a cabo la valoración de la amenaza es utilizar el método DREAD⁸. En este caso, se trata de añadir nuevas dimensiones que ayudan a determinar cuál es el impacto que representa. Con este método, se valorarán las cuestiones siguientes para cada amenaza:

⁽⁸⁾DREAD es el acrónimo de *damage potential, reproducibility, exploitability, affected users, discoverability*.

- **Daño potencial (*damage potential*)**: ¿cuál es la magnitud del daño que puede causar?
- **Reproductibilidad (*reproducibility*)**: ¿es fácil reproducir el ataque?
- **Explotabilidad (*exploitability*)**: ¿es fácil que un atacante consiga explotar una vulnerabilidad?
- **Usuarios afectados (*affected users*)**: ¿cuántos usuarios se verían afectados?
- **Detectabilidad (*discoverability*)**: ¿es fácil encontrar la vulnerabilidad?

En este tipo de valoración, se pueden utilizar los valores de alto, medio y bajo, que se cuantifican como 3, 2 y 1, respectivamente.

Ejemplo

Amenaza	D	R	E	A	D	Total	Clasificación
<i>SQL injection</i>	3	3	3	3	2	14	Alto

3.1.5. Mitigación de amenazas

El objetivo de la gestión de riesgos es reducir el impacto que puede crear la explotación de una amenaza en la aplicación y/o en el sistema. Esto se puede llevar a cabo con una estrategia de mitigación de riesgos para responder a las amenazas. En general, hay cuatro opciones para mitigar las amenazas:

- No hacer nada.
- Informar al usuario de la amenaza.

- Eliminar el problema.
- Solucionar el problema.

1) No hacer nada

Ante un riesgo bajo, la organización puede decidir adoptar esta opción. Aun así, rara vez se trata de una buena elección. No suele ser la solución correcta, puesto que el problema está latente en la aplicación y tarde o temprano será descubierto, y al final se tendrá que solucionar de todas formas.

2) Informar al usuario de la amenaza

La alternativa de informar al usuario del problema, por ejemplo con un cuadro de diálogo, permite que este pueda decidir si utilizar la función o no. Aun así, esta opción puede ser problemática, puesto que el usuario quizás tome la decisión menos acertada.

3) Eliminar el problema

Si no hay tiempo para arreglar el problema, se tiene que considerar el hecho de quitar la funcionalidad problemática de la aplicación y solucionar el problema para la próxima versión.

4) Solucionar el problema

Seleccionar las tecnologías necesarias para solucionar el problema es obviamente la mejor opción, pero también la más difícil, puesto que implica más trabajo para los diseñadores, los desarrolladores y el equipo de pruebas.

A continuación, se muestra una lista de algunas técnicas de mitigación de amenazas, utilizando la clasificación STRIDE:

Tipo de amenaza	Técnica de mitigación
<i>Spoofing</i>	Autenticación Protección de los datos confidenciales
<i>Tampering</i>	Autorización Firma digital <i>Hashes</i> <i>Tamper-resistant protocols</i>
<i>Repudiation</i>	Firma digital
<i>Information disclosure</i>	Autorización <i>Privacy-enhanced protocols</i> Cifrado
<i>Denial of service</i>	Autenticación Autorización Filtrado Calidad de servicio (QoS)

Tipo de amenaza	Técnica de mitigación
<i>Elevation of privilege</i>	Ejecución con mínimos privilegios

3.2. Herramientas

En las actividades del modelado de amenazas, se requiere definir, identificar, documentar, efectuar diagramas, hacer un seguimiento de las actividades, etc. Es decir, se necesita una gestión completa y seguir una metodología. Encontramos herramientas para facilitar la ejecución de todas las tareas y, al mismo tiempo, compartir la información con todo el equipo involucrado. En este ejemplo se muestran las distintas opciones del software SDL Threat Modeling Tool, desarrollado por Microsoft. Se trata de un software gratuito, que se puede descargar desde el portal de Microsoft y que permite llevar a cabo las distintas tareas del modelado de amenazas.

The screenshot displays the SDL Threat Modeling Tool version 3.1.8. The main window is titled "SDL Threat Modeling Tool - SDL Threat Modeling Tool v3.1.8". The left sidebar, titled "Analyze Model", contains a tree view of application components and their threats. The current node selected is "Bug status (#)? (BugPluginCmd.exe to Bug tracking system)".

The right side of the window is divided into two main sections:

- Bug status (#)?** (Top section):
 - Data Flow from "BugPluginCmd.exe" to "Bug tracking system"**
 - Subject to: Tampering, Information Disclosure, Denial Of Service
 - Do not auto generate threats for this element because [text input]
 - ⚠ Crosses boundaries: (TrustBoundary).
- Threat type: Tampering** (Bottom section):
 - Some questions to ask about this threat type:
 - Tampering is altering the bits on the wire or between processes
 - Is the dataflow timestamped/sequenced and integrity protected?
 - Do you check the dataflow for duplicate/overlapped data?
 - Are all endpoints mutually authenticated with keys obtained or validated out of band?
 - Is there a cryptographically strong message integrity system?
 - Is there a cryptographically strong channel integrity system?
 - Certify that there are no threats of this type
 - ID: Impact: 71 Solution: Someone could cause the wrong bug information to show up in the tool. On MS Corp net, IPsec. This is challenging to address generically--bug tracking systems may support different protocols.
 - Add Threat Completion: [progress bar] Finished
 - Describe the threat impact and how you will mitigate it. Bug: [File bug](#)
 - Delete Threat

Below the main sections, there are two more threat types:

- Threat type: Information Disclosure** (Bottom section):
 - Some questions to ask about this threat type:
 - Information disclosure is when the information can be read by an unauthorized party.
 - Are all endpoints mutually authenticated with keys obtained or validated out of band?
 - Is there a cryptographically strong channel confidentiality system?
 - Have you performed a side channel analysis?
 - Is there a cryptographically strong message confidentiality system?
 - Certify that there are no threats of this type
 - ID: Impact: 72 Solution: Someone with network sniffing can read the bug status request data, deriving information about bugs related to the threat model. Difficult to address systematically, depends on the plugin. (See ext. sec note #1). In many corporate network environments this could be addressed by using IPsec.
 - Add Threat Completion: [progress bar] Finished
 - Describe the threat impact and how you will mitigate it. Bug: [File bug](#)
 - Delete Threat

The bottom navigation bar includes numbered steps: 1 Draw Diagrams, 2 Analyze Model (highlighted), 3 Describe Environment, and 4 Generate Reports.

SDL Threat Modeling Tool - SDL Threat Modeling Tool v3.1.8

File Edit Actions Help

Analyze Model

All Elements

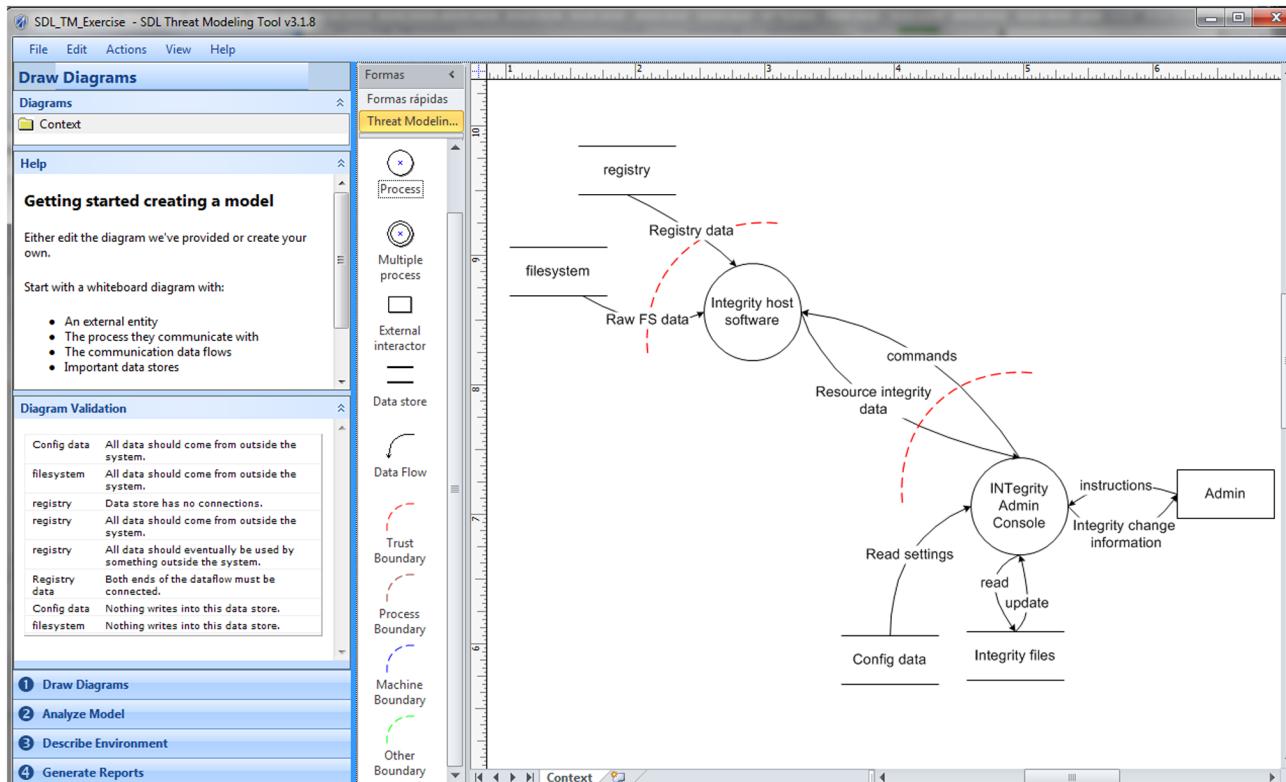
ID	Element Name	Element Type	Element Diagram References	Threat Type	Bug ID	Completion
38	Request help (SDLTM.exe to KB:help)	DataFlow	Context	InformationDisclos...	5937	<div style="width: 100%;"><div style="width: 100%;"></div></div>
39	Request help (SDLTM.exe to KB:help)	DataFlow	Context	DenialOfService	5938	<div style="width: 100%;"><div style="width: 100%;"></div></div>
40	Request update (SDLTM.exe to KB:update)	DataFlow	Context	Tampering	5939	<div style="width: 100%;"><div style="width: 100%;"></div></div>
41	Request update (SDLTM.exe to KB:update)	DataFlow	Context	InformationDisclos...	5940	<div style="width: 100%;"><div style="width: 100%;"></div></div>
42	Request update (SDLTM.exe to KB:update)	DataFlow	Context	DenialOfService	5941	<div style="width: 100%;"><div style="width: 100%;"></div></div>
6	responses (SDLTM.exe to User)	DataFlow	Context	(NotGenerated)		<div style="width: 100%;"><div style="width: 100%; background-color: blue;"></div></div>
126	run (Setup User to Setup)	DataFlow	Context	(NotGenerated)		<div style="width: 100%;"><div style="width: 100%; background-color: blue;"></div></div>
9	Save threat models (SDLTM.exe to storage)	DataFlow	Context	Tampering	5948	<div style="width: 100%;"><div style="width: 100%;"></div></div>
10	Save threat models (SDLTM.exe to storage)	DataFlow	Context	InformationDisclos...	5949	<div style="width: 100%;"><div style="width: 100%;"></div></div>
80	Save threat models (SDLTM.exe to storage)	DataFlow	Context	InformationDisclos...		<div style="width: 100%;"><div style="width: 100%;"></div></div>
11	Save threat models (SDLTM.exe to storage)	DataFlow	Context	DenialOfService	5950	<div style="width: 100%;"><div style="width: 100%;"></div></div>
29	Start process and Request bug data (SDLTM.exe to BugPl...	DataFlow	Context	Tampering	5924	<div style="width: 100%;"><div style="width: 100%;"></div></div>
30	Start process and Request bug data (SDLTM.exe to BugPl...	DataFlow	Context	InformationDisclos...	5925	<div style="width: 100%;"><div style="width: 100%;"></div></div>
31	Start process and Request bug data (SDLTM.exe to BugPl...	DataFlow	Context	DenialOfService	5926	<div style="width: 100%;"><div style="width: 100%;"></div></div>
76	State (Bug tracking system to BugPluginCmd.exe)	DataFlow	Bug Plugins	(NotGenerated)		<div style="width: 100%;"><div style="width: 100%; background-color: blue;"></div></div>
115	write (Setup to Guidance Questions, Bug Templates)	DataFlow	Context	(NotGenerated)		<div style="width: 100%;"><div style="width: 100%; background-color: blue;"></div></div>
98	Guidance Questions, Bug Templates	DataStore	Context	Tampering	5955	<div style="width: 100%;"><div style="width: 100%;"></div></div>
102	Guidance Questions, Bug Templates	DataStore	Context	Tampering		<div style="width: 100%;"><div style="width: 100%;"></div></div>
99	Guidance Questions, Bug Templates	DataStore	Context	Repudiation	5956	<div style="width: 100%;"><div style="width: 100%;"></div></div>
100	Guidance Questions, Bug Templates	DataStore	Context	InformationDisclos...	5957	<div style="width: 100%;"><div style="width: 100%;"></div></div>
101	Guidance Questions, Bug Templates	DataStore	Context	DenialOfService	5958	<div style="width: 100%;"><div style="width: 100%;"></div></div>
21	storage	DataStore	Context	Tampering	5955	<div style="width: 100%;"><div style="width: 100%;"></div></div>
81	storage	DataStore	Context	Tampering		<div style="width: 100%;"><div style="width: 100%;"></div></div>
22	storage	DataStore	Context	Repudiation	5956	<div style="width: 100%;"><div style="width: 100%;"></div></div>
23	storage	DataStore	Context	InformationDisclos...	5957	<div style="width: 100%;"><div style="width: 100%;"></div></div>
24	storage	DataStore	Context	DenialOfService	5958	<div style="width: 100%;"><div style="width: 100%;"></div></div>
74	Bug tracking system	Interactor	Bug Plugins	(NotGenerated)		<div style="width: 100%;"><div style="width: 100%; background-color: blue;"></div></div>
35	KB:help	Interactor	Context	Spoofing	5959	<div style="width: 100%;"><div style="width: 100%;"></div></div>
36	KB:help	Interactor	Context	Repudiation	5960	<div style="width: 100%;"><div style="width: 100%;"></div></div>
53	KB:update	Interactor	Context	Spoofing	5959	<div style="width: 100%;"><div style="width: 100%;"></div></div>
55	KB:update	Interactor	Context	Repudiation	5960	<div style="width: 100%;"><div style="width: 100%;"></div></div>
118	Setup User	Interactor	Context	(NotGenerated)		<div style="width: 100%;"><div style="width: 100%; background-color: blue;"></div></div>
1	User	Interactor	Context	Spoofing	5961	<div style="width: 100%;"><div style="width: 100%;"></div></div>
2	User	Interactor	Context	Repudiation	5962	<div style="width: 100%;"><div style="width: 100%;"></div></div>
62	BugPluginCmd.exe	Process	Context,Bug P...	Spoofing	5963	<div style="width: 100%;"><div style="width: 100%;"></div></div>
63	BugPluginCmd.exe	Process	Context,Bug P...	Tampering	5968	<div style="width: 100%;"><div style="width: 100%;"></div></div>
64	BugPluginCmd.exe	Process	Context,Bug P...	Repudiation	5967	<div style="width: 100%;"><div style="width: 100%;"></div></div>
65	BugPluginCmd.exe	Process	Context,Bug P...	InformationDisclos...	5966	<div style="width: 100%;"><div style="width: 100%;"></div></div>
66	BugPluginCmd.exe	Process	Context,Bug P...	DenialOfService	5965	<div style="width: 100%;"><div style="width: 100%;"></div></div>
67	BugPluginCmd.exe	Process	Context,Bug P...	ElevationOfPrivilege	5964	<div style="width: 100%;"><div style="width: 100%;"></div></div>
56	SDLTM.exe	Process	Context	Spoofing	5963	<div style="width: 100%;"><div style="width: 100%;"></div></div>
57	SDLTM.exe	Process	Context	Tampering	5968	<div style="width: 100%;"><div style="width: 100%;"></div></div>

① Draw Diagrams

② Analyze Model

③ Describe Environment

④ Generate Reports



4. Técnicas de seguridad

En este apartado se definen algunas de las técnicas de seguridad que se pueden aplicar a los sistemas de información. Se trata de mostrar un breve resumen de cada tecnología, sin entrar en profundidad en cada una.

1) Autenticación

La autenticación es el proceso por el que una entidad verifica que otra es la que dice ser. Hay que tener en cuenta que una entidad puede ser un usuario, código ejecutable o un ordenador. La autenticación requiere evidencia en forma de credenciales y pruebas, que pueden ser de muchas formas, como una contraseña, una clave privada o, en el caso de la autenticación biométrica, una huella dactilar.

2) Autorización

Una vez una entidad se ha autenticado, en general querrá acceder a los recursos que le permite la aplicación, como por ejemplo, datos, ficheros, impresoras, funcionalidades, etc. En una misma aplicación, muchas veces no todos los usuarios tienen el mismo nivel de privilegios y, por lo tanto, la aplicación debe comprobar si el usuario está autorizado para llevar a cabo una tarea o acceder a un recurso determinado. Una buena alternativa es que, dependiendo del usuario, la aplicación muestre solo las opciones y los recursos en función de los privilegios del usuario y este solo pueda acceder a lo que está autorizado.

Algunos de los mecanismos de autorización son los siguientes:

- Listas de control de acceso (ACL).
- Restricciones de IP.
- Permisos de servidor.

a) Listas de control de acceso

La implementación de una ACL (*access control list*) se basa en una lista de permisos que se asocian a un objeto o recurso (datos, carpetas, ficheros, opciones, etc.) en función del usuario o grupo de usuarios. En general, los sistemas operativos implementan este concepto y, en función de la aplicación, será necesario valorar si es mejor crear una lista de control de acceso para la propia aplicación o utilizar la que proporciona el sistema operativo.

b) Restricciones de IP

En este caso se trata de dar o denegar acceso a los recursos en función de la dirección IP, de las subredes o del DNS.

c) Permisos de servidor

Muchos servidores ofrecen su propia forma de control de acceso para proteger sus objetos. Por ejemplo, los servidores de bases de datos permiten al administrador determinar quién tiene acceso a las tablas, funciones, procedimientos o vistas.

3) *Tamper-resistant and privacy-enhanced technologies*

La protección de la manipulación y la privacidad se refiere a la capacidad de proteger los datos de una manipulación, ya sea maliciosa o accidental.

Algunos de los protocolos y las tecnologías que permiten esta protección son los siguientes:

- SSL/TLS.
- IPSec.
- DCOM y RPC.
- Sistemas de ficheros cifrados.

a) SSL/TLS

Se trata de protocolos criptográficos que proporcionan comunicaciones seguras a través de la Red. Los datos son cifrados con un código de autenticación del mensaje para proporcionarles integridad. TLS es la versión de SSL ratificada por la Internet Engineering Task Force (IETF).

b) IPSec

Su función es la de asegurar las comunicaciones sobre el protocolo IP autenticando y/o cifrando cada paquete IP.

c) DCOM

Distributed component object model (DCOM) es una tecnología propietaria de Microsoft para desarrollar componentes de software distribuido entre varios ordenadores que se comunican entre sí. La aparición del entorno de trabajo .NET por parte de Microsoft hizo que las nuevas aplicaciones en entorno Windows dejaseen de usarlo. Aun así, muchas implementaciones todavía utilizan DCOM.

d) Sistemas de ficheros cifrados

Son sistemas de ficheros especializados en cifrado, que permiten a los usuarios almacenar datos cifrados en el disco duro de manera transparente. Algunos de los sistemas de ficheros cifrados son los siguientes:

- EFS en el caso de Microsoft Windows Server, que opera en un nivel de sistema de ficheros.
- CFS en UNIX, que opera en un nivel de aplicación.
- TCFS en Linux, que opera en un nivel de núcleo.

4) Firma digital

La firma digital es una aplicación de la criptografía de clave pública que permite dar autenticidad de origen a la información enviada, asegurar su integridad e impedir el repudio de quien firma.

Las propiedades de la firma digital son las siguientes:

- Autenticidad: la firma autentica el documento.
- No repudio: quien ha firmado un documento no puede negar que lo ha firmado.
- Integridad: el contenido del mensaje junto con la clave privada se utiliza para crear la firma. En caso de modificación posterior del mensaje, la verificación de la firma con el nuevo mensaje modificado no será correcta.

A continuación, se muestra cómo opera una firma digital.

Una función **hash** o función **resumen** hace corresponder a un mensaje m de tamaño variable una representación $H(m)$ de tamaño fijo.

Los algoritmos más utilizados son MD5 y SHA-1.

Propiedades:

- Soportan entradas de tamaño variable.
- Longitud fija del resumen generado.
- Función de un único sentido; es decir, a partir del resumen generado no se puede deducir la entrada original.
- Función con distribución uniforme de las colisiones (una colisión se produce cuando dos entradas diferentes generan el mismo resumen).

Firmar:

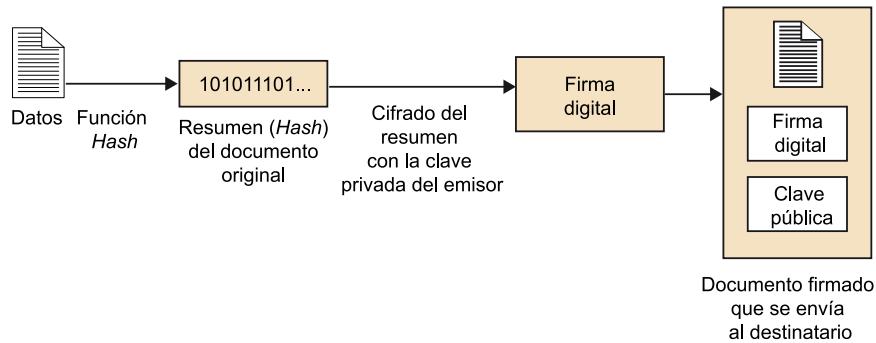
Para firmar un mensaje m , el usuario A aplica la función $hashH$ al mensaje m y obtiene $H(m)$.

Firma $H(m)$ con su clave privada SK y obtiene la firma $s(m)$ donde $s(m) = SK(H(m))$.

Se añade $s(m)$ a m , y se forma la firma.

Se añade la clave pública PK del usuario A , para que el destinatario pueda comprobar la firma.

Se envía al destinatario el paquete formado por $(m, s(m), PK)$.

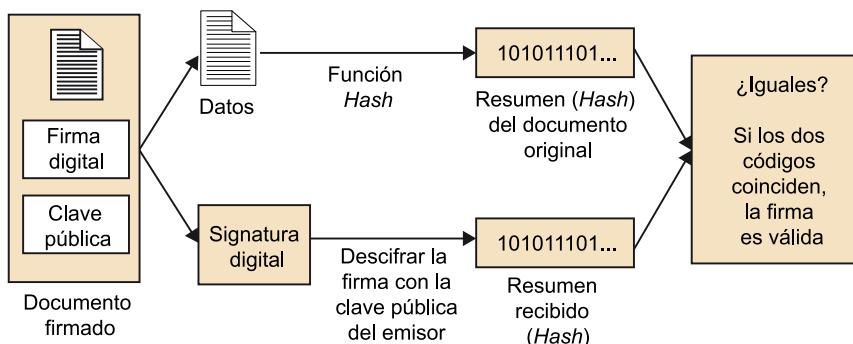


Verificar:

El usuario B recibe el mensaje de A y calcula $H(m)$, de modo que obtiene el resumen 1.

Descifra $s(m)$ con la clave pública de A y obtiene el resumen 2.

Se comparan los resúmenes 1 y 2; si son iguales, entonces el documento es el original firmado. Puesto que las acciones de PK y SK son inversas, el verificador podrá comprobar que el resultado es exactamente el mensaje firmado.



5) Auditoría

El objetivo de la auditoría es recoger información sobre el comportamiento, el acceso o la denegación de acceso a los recursos y los objetos, el uso de privilegios y otras acciones de seguridad importantes. Esta información se almacena en un fichero de registros (fichero *log*) para su análisis posterior.

Es importante proteger estos ficheros contra los posibles ataques, puesto que el atacante podría obtener información muy valiosa para efectuar sus ataques.

6) Filtrado y calidad de servicio

El filtrado implica inspeccionar los datos a medida que se reciben, para tomar la decisión de aceptar o rechazar el paquete. Esta es una de las funcionalidades de los cortafuegos, y muchas amenazas se pueden mitigar por medio de este filtrado.

La calidad del servicio (*Quality of Service*) es un conjunto de componentes que permiten proporcionar un tratamiento específico a determinados tipos de tráfico.

7) Mínimos privilegios

Una buena práctica es la de ejecutar los procesos justo con los privilegios necesarios e imprescindibles para llevar a cabo la tarea en cuestión.

Bibliografía

Appel Inc. (2013). *Risk Assessment and Threat Modeling* [en línea]. https://developer.apple.com/library/mac/#documentation/security/Conceptual/Security_Overview/ThreatModeling/ThreatModeling.html

Howard, M; LeBlanc, D. (2002). *Writing Secure Code, Second Edition*. Redmond Washington: Microsoft Press.

Meier, J. D.; Mackman, A. (2005). *Security Engineering Explained, patterns & practices*. Microsoft.

Microsoft (2013). *Microsoft Security Development Lifecycle (SDL) Process Guidance* [en línea]. <http://msdn.microsoft.com>

Microsoft (2013). *Security and Identity* [en línea]. <http://msdn.microsoft.com>

Microsoft (2013). *Threat Modeling* [en línea]. <http://msdn.microsoft.com>

OWASP. The Open Web Application Security Project (2013). *Application Threat Modeling* [en línea]. https://www.owasp.org/index.php?title=Application_Threat_Modeling&setlang=en

OWASP. The Open Web Application Security Project (2013). *Security Code Review in the SDL* [en línea]. https://www.owasp.org/index.php/Security_Code_Review_in_the_SDLC

OWASP. The Open Web Application Security Project (2013). *Threat Risk Modeling* [en línea]. https://www.owasp.org/index.php/Threat_Risk_Modeling

