




# Ingeniería de Software II

## UNIDAD I. Calidad

### CAPÍTULO 4: SEGURIDAD EN LA INGENIERÍA DE SOFTWARE



Detente y echa un vistazo a tu alrededor. **¿Dónde crees que se está implementando el software?** Seguro está en tu computadora portátil, tableta y teléfono celular. ¿Qué pasa con tus electrodomésticos, refrigerador, lavavajillas, etc.? ¿Y tu coche?

**Transacciones financieras: cajeros automáticos**, ¿Banca en línea, software financiero, software fiscal? ¿Su proveedor de electricidad? Definitivamente usando software. ¿Tienes algún dispositivo wearable puesto? ¿Un fit-bit? Quizás Tiene dispositivos médicos, como un marcapasos. La conclusión es: el software lo es todo a nuestro alrededor, sobre nosotros y, a veces, en nosotros.

**Cada producto de software tiene el potencial para ser hackeado**, a veces con consecuencias nefastas. Esta es la razón por la que nosotros, como los ingenieros de software deben preocuparse por la seguridad del software.

*Contribución de: Nancy Mead Universidad Carnegie Mellon Instituto de Ingeniería de Software*


# Panorama general



*¿Qué es? La ingeniería de seguridad de software es un conjunto de técnicas que pueden mejorar la seguridad del software mientras está en desarrollo.*

*¿Quién lo hace? Aunque los ingenieros de software no necesitan convertirse en expertos en seguridad, necesitan colaborar con expertos en seguridad.* Los expertos en seguridad pueden ser miembros del equipo de trabajo, en un equipo especializado separado, o pueden ser consultores externos.

*¿Por qué es importante? Los medios de comunicación informan continuamente sobre casos de hacking, ya sea por parte de gánsters, competidores corporativos, naciones hostiles, o cualquier otro actor malintencionado. Las consecuencias para infraestructuras críticas, instituciones financieras, el cuidado de la salud y todos los aspectos de la vida moderna son significativos.*



**¿Cuáles son los pasos?** Hay una serie de medidas que se pueden tomar para garantizar que el software es seguro. Discutiremos algunos de ellos aquí y proporcionar indicaciones a los recursos para obtener más información exploración.

**¿Cuál es el producto del trabajo?** Como verás, hay muchos productos de trabajo que se desarrollan en el proceso de ingeniería de software seguro. El producto de trabajo final, o en curso, es el software que se ha desarrollado utilizando prácticas seguras de ingeniería de software.


**¿Cómo me aseguro de que lo he hecho bien?** Todo lo que discutiremos como método para mejorar la seguridad del software, ya sea a nivel organizacional o de proyecto, puede y debe ser revisada por las partes interesadas. Además, los procesos de desarrollo seguros pueden ser mejorados, si se comprueba que faltan.



## 4.1 Por qué la ingeniería de seguridad del software es importante

La seguridad del software es algo más que proteger el software operativo con firewalls, contraseñas seguras y encriptación.

También se trata de desarrollar software de tal manera que es más seguro desde el principio. Ya disponemos de técnicas que nos ayudarán a desarrollar software que sea significativamente más seguro de lo que sería de otra manera.



En este capítulo, vamos a ver algunos de los modelos y técnicas que pueden ayudarnos a lograr una mejor seguridad del software. Comenzaremos analizando el proceso de modelos de seguridad. A continuación, examinaremos las actividades específicas del proceso, incluidas actividades como ingeniería de requisitos, casos de uso indebido o abuso, análisis de riesgos de seguridad, modelado de amenazas, superficie de ataque, codificación segura y medición.

También tendremos en cuenta la seguridad Modelos de mejora de procesos. Finalmente, resumiremos y le proporcionaremos una lista de referencias para que puedas profundizar en cualquiera de estos temas.

La investigación en ingeniería de seguridad de software es un área muy activa. En este libro estamos solo proporciona una visión general de los métodos y herramientas para respaldar la práctica real. Hay muchos libros (por ejemplo, [Mea16], [Shu13] y [Hel18]) y otros recursos dedicados exclusivamente a la ingeniería de seguridad de software, y te indicaremos algunas de ellas. 390



## 4.2 Modelos de ciclo de vida de seguridad

*El Ciclo de Vida de Desarrollo de Seguridad (SDL) de Microsoft [Mea16] [Mic18] es un proceso de seguridad de software líder en la industria.* Una iniciativa de Microsoft y una política obligatoria. Desde 2004, el SDL ha permitido a Microsoft integrar la seguridad y la privacidad en su software y cultura.

El SDL introduce la seguridad y la privacidad desde el principio y a lo largo de todas las fases del proceso de desarrollo, y es, sin duda, el modelo de ciclo de vida de desarrollo de seguridad más conocido y utilizado.

Microsoft ha definido un conjunto de principios denominados *Seguridad por Diseño, Seguridad por Defecto, Seguridad en la Implementación y Comunicaciones (SD3+C)* para ayudar a determinar dónde se necesitan medidas de seguridad. Estos son los siguientes [Mic10]:

# Seguridad por Diseño



**Arquitectura, diseño y estructura seguros.** Los desarrolladores consideran los problemas de seguridad como parte del diseño arquitectónico básico del desarrollo de software. Revisan diseños detallados para detectar posibles problemas de seguridad y diseñan y desarrollan mitigaciones para todas las amenazas.

**Modelado y mitigación de amenazas.** Se crean modelos de amenazas y las mitigaciones de amenazas están presentes en todas las especificaciones de diseño y funcionales.

**Eliminación de vulnerabilidades.** Tras la revisión, no quedan en el código vulnerabilidades de seguridad conocidas que representen un riesgo significativo para el uso previsto del software. Esta revisión incluye el uso de herramientas de análisis y pruebas para eliminar las diferentes clases de vulnerabilidades.

**Mejoras en la seguridad.** Se desestiman los protocolos y códigos heredados menos seguros y, siempre que sea posible, se ofrecen a los usuarios alternativas seguras que cumplen con los estándares del sector.






# Seguridad por defecto

**Mínimo privilegio.** Todos los componentes se ejecutan con la menor cantidad de permisos posible.

**Defensa exhaustiva.** Los componentes no dependen de una única solución de mitigación de amenazas que exponga a los usuarios si falla.

**Configuración predeterminada conservadora.** El equipo de desarrollo conoce la superficie de ataque del producto y la minimiza en la configuración predeterminada.



**Evita cambios predeterminados riesgosos.** Las aplicaciones no realizan cambios predeterminados en el sistema operativo ni en la configuración de seguridad que reduzcan la seguridad del equipo host. En algunos casos, como en el caso de los productos de seguridad, es aceptable que un programa de software fortalezca (aumente) la configuración de seguridad del equipo host.

Las infracciones más comunes de este principio son los juegos que abren puertos de firewall sin informar al usuario o que les indican que abran puertos de firewall sin informarles de los posibles riesgos.

**Servicios menos utilizados desactivados por defecto.** Si menos del 80 % de los usuarios de un programa utiliza una función, esta no debería activarse por defecto. Medir el 80 % del uso de un producto suele ser difícil, ya que los programas están diseñados para muchos perfiles diferentes (por ejemplo, principiantes, expertos, técnicos, etc.). Puede ser útil considerar si una función aborda un escenario de uso principal para todos los perfiles. De ser así, a veces se denomina función P1. ( ***“¿Esta función es fundamental para la mayoría de los usuarios, sin importar su perfil?” Si la respuesta es sí, entonces esa función puede considerarse una función clave o una función P1 (Prioridad 1), y en ese caso sí debería activarse por defecto.*** )



# Seguridad en la Implementación

**Guías de implementación.** Las guías de implementación prescriptivas describen cómo implementar cada función de un programa de forma segura, incluyendo información que permite a los usuarios evaluar el riesgo de seguridad al activar opciones no predeterminadas (y, por lo tanto, aumentar la superficie de ataque).

**Herramientas de análisis y gestión.** Las herramientas de análisis y gestión de seguridad permiten a los administradores determinar y configurar el nivel de seguridad óptimo para una versión de software.

**Herramientas de implementación de parches.** Las herramientas de implementación facilitan la implementación de parches **(su propósito es ayudar a instalar parches (actualizaciones o correcciones) en otros programas o sistemas de forma más fácil y rápida).**

# Comunicaciones

**Respuesta de seguridad.** Los equipos de desarrollo responden con prontitud a los informes de vulnerabilidades de seguridad y comunican información sobre actualizaciones de seguridad.

**Participación de la comunidad.** Los equipos de desarrollo interactúan proactivamente con los usuarios para responder preguntas sobre vulnerabilidades de seguridad, actualizaciones de seguridad o cambios en el panorama de seguridad.



Mgt. Ing. Willian Zamalloa Paro



Agencia Nacional de Ciberseguridad (ANCI)  
@ANCiChile

...

Indisponibilidad provocada por [#CrowdStrike](#) - Comunicado | El problema se debe a un error en una actualización de CrowdStrike instalada durante la noche, no es un ciberataque, ni tampoco una falla en [#Windows](#). Más info: [csirt.gob.cl/alertas/cnd24-...](#)

## Alerta de Incidentes



6:59 a. m. · 19 jul. 2024 · 27,7 mil Visualizaciones



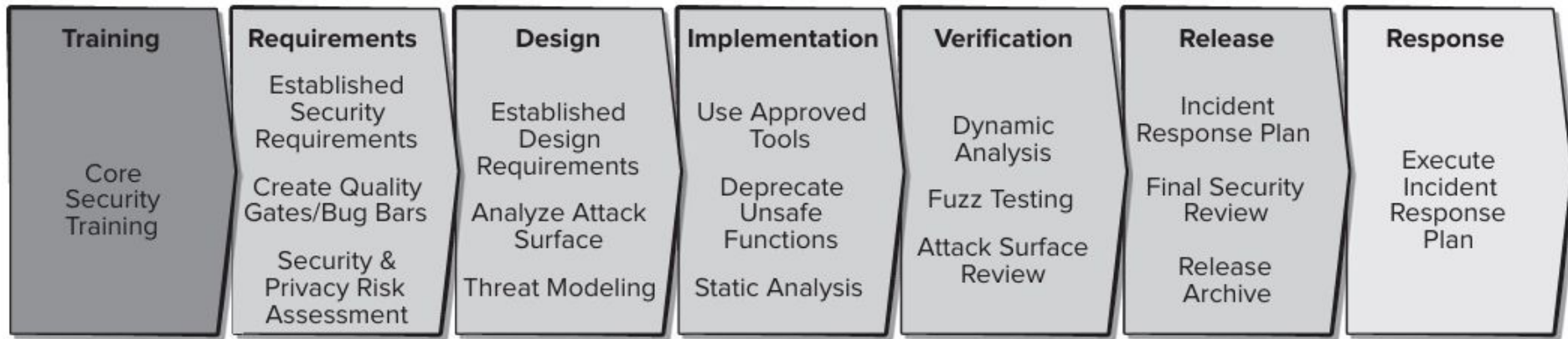
El modelo del proceso de desarrollo de software seguro se asemeja al que se muestra en la Figura 18.1.

La documentación de Microsoft SDL describe lo que deben hacer los arquitectos, diseñadores, desarrolladores y evaluadores para cada una de las 16 prácticas recomendadas. Los datos recopilados por Microsoft tras la implementación de SDL muestran una reducción significativa de las vulnerabilidades, lo que se tradujo en la necesidad de menos parches y, por consiguiente, en un ahorro significativo de costes. Le recomendamos que visite el sitio web de SDL para obtener más información sobre estas prácticas. Desde el desarrollo de SDL, se han publicado numerosos artículos, libros, cursos de formación, etc., que complementan el modelo SDL.<sup>1</sup>

<sup>1</sup> Más recientemente, Microsoft ha demostrado cómo las actividades de SDL pueden integrarse con un enfoque de desarrollo ágil: <https://www.microsoft.com/en-us/securityengineering/sdl/>

**FIGURE 18.1****Secure Software Development Process Model at Microsoft**

Adapted from Shunn, A., et al. Strengths in Security Solutions, Software Engineering Institute, Carnegie Mellon University, 2013. Available at <http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=77878>.



[https://insights.sei.cmu.edu/documents/376/2013\\_019\\_001\\_77887.pdf](https://insights.sei.cmu.edu/documents/376/2013_019_001_77887.pdf)

## 4.3 Actividades del ciclo de vida del desarrollo seguro

Un enfoque diferente, independiente del modelo de ciclo de vida, es el de los puntos de contacto para la seguridad del software [McG06]. Este enfoque argumenta que lo que importa son las actividades (puntos de contacto), no el modelo. Las actividades pueden incorporarse a cualquier modelo de ciclo de vida y, por lo tanto, se consideran independientes del proceso.

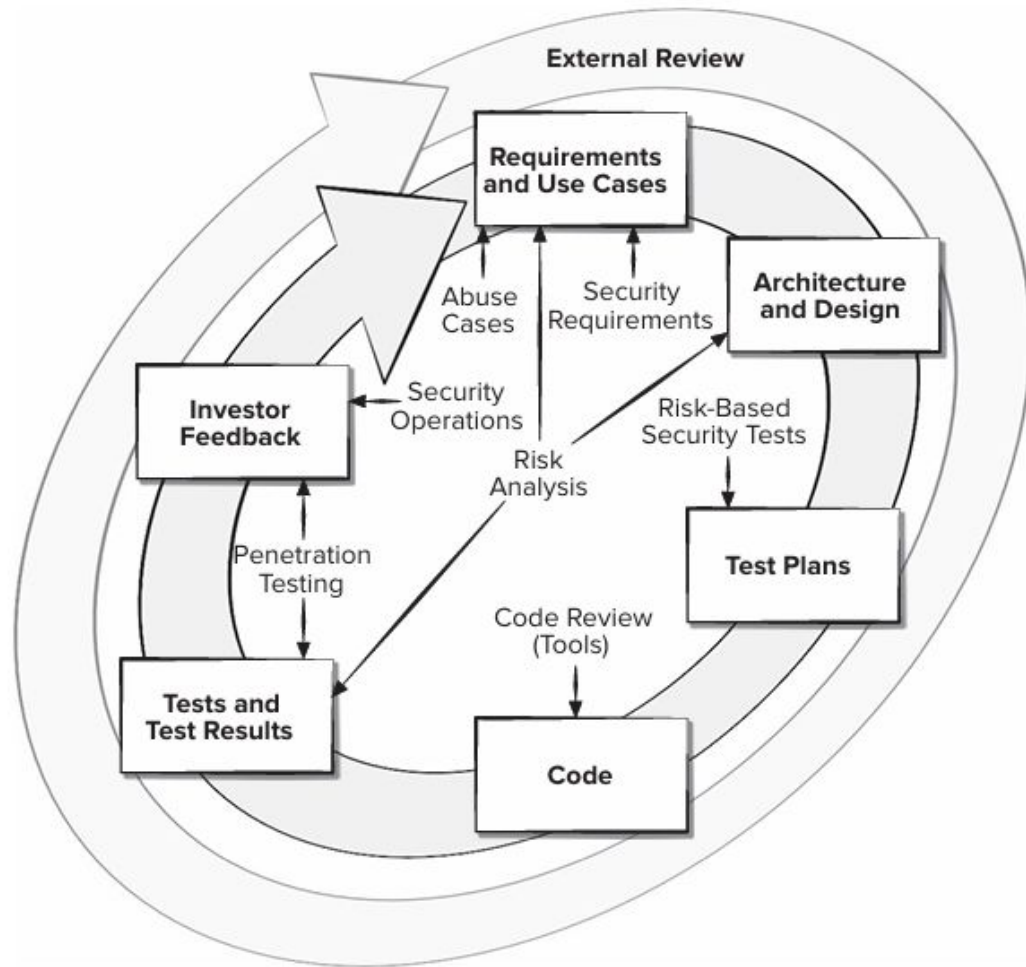
Posteriormente, los puntos de contacto formaron la base de BSIMM (Building Security In Maturity Model), un modelo de madurez que se analizará más adelante en este capítulo. Algunas organizaciones consideran que los puntos de contacto son el conjunto mínimo de actividades que deben realizarse en el desarrollo de software seguro.

La Figura 18.2 muestra una representación gráfica de los puntos de contacto. En este diagrama, las actividades de seguridad recomendadas aparecen encima de la actividad de desarrollo de software correspondiente, o fase del ciclo de vida:

Con el SDL y los puntos de contacto en mente, analizaremos algunas de las actividades importantes de desarrollo de software de seguridad asociadas a ellos.

**FIGURE 18.2**

**Software  
security  
touchpoints**







## 4.4 Ingeniería de requisitos de seguridad

Aunque los requisitos de seguridad son una parte importante del desarrollo de software seguro, en la práctica suelen descuidarse. Cuando existen, suelen ser un complemento, copiado de una lista genérica de características de seguridad. *La ingeniería de requisitos necesaria para obtener un mejor conjunto de requisitos de seguridad rara vez se lleva a cabo [All08].*

La práctica de la ingeniería de requisitos suele abordar las características deseadas por el usuario. Por lo tanto, se presta atención a la funcionalidad del sistema desde la perspectiva del usuario, pero se presta poca atención a lo que el sistema no debería hacer [Bis02]. *Los usuarios esperan que los sistemas sean seguros, y estas suposiciones deben incorporarse a los requisitos de seguridad de los sistemas de software antes de su desarrollo, no después. A menudo, las suposiciones de los usuarios sobre la seguridad se ignoran porque las características del sistema son el enfoque principal.*



Además de los modelos de ciclo de vida de la seguridad, existen muchos modelos de proceso específicos para los requisitos de seguridad.


Estos incluyen: artefactos de requisitos de seguridad básicos [Mof04], Reducción de Costos de Software (SCR) [Hei02], **SQUARE (Ingeniería de Requisitos de Calidad de Seguridad) (Security QUALity Requirements Engineering)**[Mea05] y Proceso de Ingeniería de Requisitos de Seguridad (SREP) (Security Requirements Engineering Process )[Mel06]. En el resto de esta sección, consideraremos SQUARE como un ejemplo representativo de los modelos de ciclo de vida de la seguridad.



## 4.4.1 SQUARE

SQUARE es un modelo representativo del proceso de ingeniería de requisitos de seguridad, pero es importante tener en cuenta que si ya cuenta con un modelo de proceso de desarrollo, como los tradicionales o actuales, puede simplemente retomar algunos de los pasos de SQUARE para mejorar su modelo existente.

*No es necesario desarrollar un proceso completamente nuevo para abordar la seguridad en sus actividades de desarrollo de software. Le sugerimos que añada definiciones de seguridad a su glosario; realice análisis de riesgos, incluyendo la identificación de posibles ataques mediante casos de uso indebido o modelado de amenazas; desarrolle estrategias de mitigación; y categorice y priorice los requisitos de seguridad candidatos.*



*El modelo de proceso SQUARE permite obtener, categorizar y priorizar los requisitos de seguridad para sistemas con uso intensivo de software. Su objetivo es integrar conceptos de seguridad en las primeras etapas del ciclo de vida del desarrollo.*

También se puede utilizar para sistemas en desarrollo y aquellos en proceso de mejora y modificación. El proceso se muestra en la Tabla 18.1, seguido de breves descripciones de cada paso.

**TABLE 18.1** The SQUARE process

Number	Step	Input	Techniques	Participants	Output
1	Agree on definitions.	Candidate definitions from IEEE and other standards	Structured interviews, focus group	Stakeholders, requirements team	Agreed-to definitions
2	Identify assets and security goals.	Definitions, candidate assets and goals, business drivers, policies and procedures, examples	Facilitated work session, surveys, interviews	Stakeholders, requirements engineer	Assets and security goals
3	Develop artifacts to support security requirements definition.	Potential artifacts (e.g., scenarios, misuse cases, templates, forms)	Work session	Requirements engineer	Needed artifacts: scenarios, misuse cases, models, templates, forms
4	Perform (security) risk assessment.	Misuse cases, scenarios, security goals	Risk assessment method, analysis of anticipated risk against organizational risk tolerance, including threat analysis	Requirements engineer, risk expert, stakeholders	Risk assessment results
5	Select elicitation techniques.	Goals, definitions, candidate techniques, expertise of stakeholders, organizational style, culture, level of security needed, cost-benefit analysis, etc.	Work session	Requirements engineer	Selected elicitation techniques
6	Elicit security requirements.	Artifacts, risk assessment results, selected techniques	Accelerated Requirements Method, Joint Application Development, interviews, surveys, model-based analysis, checklists, lists of reusable requirements types, document reviews	Stakeholders facilitated by requirements engineer	Initial cut at security requirements
7	Categorize requirements as to level (system, software, etc.) and whether they are requirements or other kinds of constraints.	Initial requirements, architecture	Work session using a standard set of categories	Requirements engineer, other specialists as needed	Categorized requirements
8	Prioritize requirements.	Categorized requirements and risk assessment results	Prioritization methods such as Analytical Hierarchy Process (AHP), Triage, Win-Win, etc.	Stakeholders facilitated by requirements engineer	Prioritized requirements
9	Inspect requirements.	Prioritized requirements, candidate formal inspection technique	Inspection method such as Fagan and peer reviews	Inspection team	Initial selected requirements, documentation of decision-making process and rationale



## 4.4.2 El proceso SQUARE

El proceso SQUARE se aplica mejor cuando lo implementan los ingenieros de requisitos del proyecto y los expertos en seguridad, con el apoyo de la dirección ejecutiva y las partes interesadas. Analicemos los pasos.<sup>2</sup>



## Paso 1. Acordar las definiciones.

Para evitar confusiones semánticas, este paso es necesario como prerequisite para la ingeniería de requisitos de seguridad. En un proyecto determinado, los miembros del equipo suelen tener definiciones en mente, basadas en su experiencia previa, pero estas definiciones suelen ser diferentes entre sí [Woo05].

Fuentes como el Instituto de Ingenieros Eléctricos y Electrónicos (IEEE) y el Cuerpo de Conocimientos de Ingeniería de Software (SWEBOK) ofrecen diversas definiciones para seleccionar o adaptar [SWE14].



## Paso 2. Identificar los activos y los objetivos de seguridad.

Este paso se realiza a nivel organizativo del proyecto y es necesario para respaldar el desarrollo de software. Las diferentes partes interesadas suelen tener inquietudes sobre diferentes activos y, por lo tanto, diferentes objetivos.

Por ejemplo, una parte interesada en recursos humanos puede preocuparse por mantener la confidencialidad de los registros del personal, mientras que una parte interesada en un área de investigación puede preocuparse por garantizar que la información del proyecto de investigación no sea accedida, modificada ni robada.





## Paso 3. Desarrollar artefactos.

Este paso es necesario para respaldar todas las actividades posteriores de ingeniería de requisitos de seguridad.

A menudo, las organizaciones no cuentan con los documentos clave necesarios para respaldar la definición de requisitos, o pueden no estar actualizados. Esto significa que se puede dedicar mucho tiempo a buscar documentos, o el equipo tendrá que actualizarlos antes de continuar.



## Paso 4. Realice una evaluación de riesgos.

Este paso requiere un experto en métodos de evaluación de riesgos, el apoyo de las partes interesadas y el de un ingeniero de requisitos de seguridad.

Existen diversos métodos de evaluación de riesgos, pero independientemente del que elija, sus resultados pueden ayudar a identificar las exposiciones de seguridad más prioritarias.



## Paso 5. Seleccionar la técnica de elicitación.

Este paso cobra importancia cuando hay diversas partes interesadas. Una técnica de elicitación más formal, como el Método de Requisitos Acelerados [Hub99], el Diseño Conjunto de Aplicaciones [Woo89] o las entrevistas estructuradas, puede ser eficaz para resolver problemas de comunicación cuando hay partes interesadas con diferentes trasfondos culturales. En otros casos, la elicitación puede consistir simplemente en reunirse con una parte interesada principal para intentar comprender sus necesidades de seguridad.



## Paso 6. Obtener los requisitos de seguridad

Este paso abarca el proceso de obtención de datos mediante la técnica seleccionada. La mayoría de las técnicas de obtención de datos proporcionan una guía detallada sobre cómo realizarla. Esto se basa en los artefactos desarrollados en pasos anteriores.



## Paso 7. Categorizar los requisitos.

Este paso permite al ingeniero de requisitos de seguridad distinguir entre los requisitos esenciales, los objetivos (requisitos deseados) y las posibles restricciones arquitectónicas. Esta categorización también facilita la priorización posterior.



## Paso 8. Priorizar los requisitos.

Este paso depende del anterior y también puede implicar un análisis de costo-beneficio para determinar qué requisitos de seguridad ofrecen una alta rentabilidad en relación con su costo. Por supuesto, la priorización también puede depender de otras consecuencias de las brechas de seguridad, como la pérdida de vidas, la pérdida de reputación y la pérdida de confianza del consumidor.



## Paso 9. Inspección de requisitos.

Esta actividad de revisión puede llevarse a cabo con distintos niveles de formalidad, como se explica en el Capítulo 2. Una vez finalizada la inspección, el equipo del proyecto debe contar con un conjunto inicial de requisitos de seguridad priorizados que puedan revisarse según sea necesario más adelante en el proyecto.

## 4.5 Casos de uso indebido y abuso y patrones de ataque


Los casos de uso indebido (o abuso) pueden ayudarle a ver su software de la misma manera que lo hacen los atacantes. Al pensar en los eventos negativos, podrá comprender mejor cómo desarrollar software seguro. Un caso de uso indebido puede considerarse un caso de uso iniciado por el atacante.

Uno de los objetivos de los casos de uso indebido [Sin00] es decidir de antemano cómo debe reaccionar el software ante posibles ataques. También puede combinar casos de uso indebido y casos de uso normales para realizar análisis de amenazas y peligros [Ale03].

Sugerimos crear casos de uso indebido mediante una lluvia de ideas. Colaborar con expertos en seguridad y expertos en la materia (SMEs) permite cubrir muchos temas rápidamente. Durante la lluvia de ideas, los expertos en seguridad de software formulan numerosas preguntas a los desarrolladores para ayudar a identificar los puntos débiles del sistema.

*Esto implica una revisión minuciosa de todas las interfaces de usuario y considera los eventos que los desarrolladores asumen que no pueden ocurrir, pero que los atacantes sí pueden provocar.*





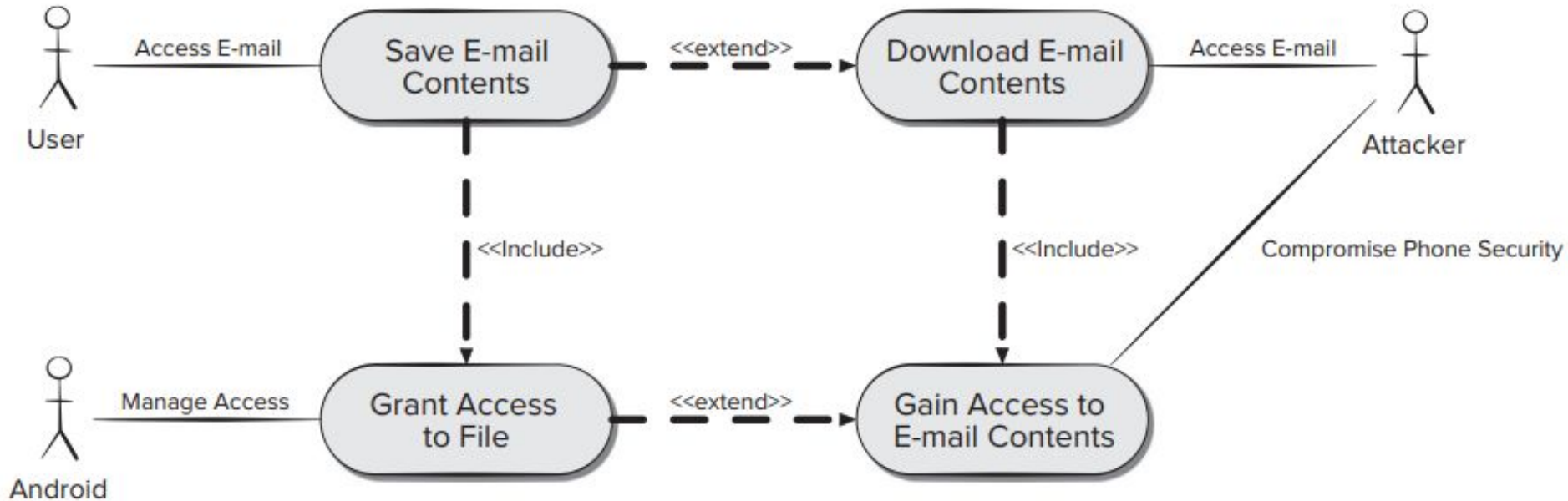
Aquí hay algunas preguntas que deben considerarse: *¿Cómo puede el sistema distinguir entre datos de entrada válidos e inválidos?* *¿Puede determinar si una solicitud proviene de una aplicación legítima o fraudulenta?* *¿Puede un usuario interno provocar un mal funcionamiento del sistema?*


Intentar responder a estas preguntas ayuda a los desarrolladores a analizar sus suposiciones y les permite solucionar los problemas con antelación. Los casos de uso indebido pueden presentarse en forma de tabla o diagrama.

La Figura 18.3 proporciona un ejemplo de caso de uso indebido que muestra cómo el malware DroidCleaner puede atacar con éxito un teléfono móvil utilizando una aplicación de correo electrónico de código abierto llamada K-9. Este ejemplo se extrae de un informe mucho más extenso que quizás le interese [Ali14].

**FIGURE 18.3**

**Misuse case (exploited by DroidCleaner): Data in an e-mail stored on the smartphone is stolen**






En este caso de uso indebido, el usuario guarda el correo electrónico en el almacenamiento externo del teléfono. El atacante accede al almacenamiento del teléfono comprometiendo el sistema operativo. Una forma común de acceder al teléfono es engañando al usuario para que instale un troyano, al cual, sin saberlo, le otorga acceso a la unidad durante el proceso de instalación. El atacante puede entonces usar el troyano para descargar archivos, incluido el contenido del correo electrónico.

Los patrones de ataque pueden ser útiles, ya que proporcionan un modelo para crear un ataque. Por ejemplo, el desbordamiento de búfer es un tipo de explotación de seguridad. Los atacantes que intentan aprovecharse de un desbordamiento de búfer utilizan pasos similares [OWA16].

Los patrones de ataque pueden documentar estos pasos (por ejemplo, tiempo, recursos, técnicas), así como las prácticas que los desarrolladores de software pueden utilizar para prevenir o mitigar su éxito [Hog04]. Al desarrollar casos de uso indebido y abuso, los patrones de ataque pueden ser útiles.



Los casos de uso indebido deben priorizarse a medida que se generan. Además, deben lograr el equilibrio adecuado entre costo y beneficio. ***El presupuesto del proyecto podría no permitir que un equipo de software implemente todas las estrategias de mitigación definidas a la vez.*** En tales casos, las estrategias pueden priorizarse e implementarse gradualmente. El equipo también puede excluir ciertos casos por considerarlos extremadamente improbables.

Existen plantillas para casos de uso indebido y abuso en diversas referencias. Pueden ser texto o diagramas y pueden estar respaldadas por herramientas. Se pueden encontrar buenas fuentes de plantillas en los materiales de Sindre y Opdahl [Sin01] y Alexander [Ale02].



## 4.6 Análisis de riesgos de seguridad

Se ha propuesto una amplia variedad de métodos de evaluación de riesgos de seguridad. Ejemplos típicos incluyen el método de Análisis de Riesgos de Ingeniería de Seguridad (SERA) de SEI CERT<sup>3</sup> y el Marco de Gestión de Riesgos (RMF) del NIST<sup>4</sup>. El RMF se ha convertido en un enfoque ampliamente utilizado que proporciona directrices para los usuarios. Los pasos del RMF para la seguridad son:


- **Categorizar** el sistema de información y la información procesada, almacenada y transmitida por dicho sistema con base en un análisis de impacto.

3 See <https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=485410>

4 See <https://csrc.nist.gov/publications/detail/sp/800-37/rev-1/final>



- **Seleccionar** un conjunto inicial de controles de seguridad básicos para el sistema de información, basándose en la categorización de seguridad; utilizando una evaluación organizacional de riesgos y las condiciones locales, adaptar y complementar el conjunto de controles de seguridad según sea necesario.
- **Implementar** los controles de seguridad y describir cómo se emplean en el sistema de información y su entorno operativo.
- **Evaluar** los controles de seguridad mediante procedimientos de evaluación adecuados para determinar en qué medida se implementan correctamente, funcionan según lo previsto y producen el resultado deseado en cuanto al cumplimiento de los requisitos de seguridad del sistema.

- 
- **Autorizar** la operación del sistema de información basándose en la determinación del riesgo para las operaciones y activos de la organización, las personas u otras organizaciones (incluida la defensa nacional), derivado de la operación del sistema de información, siempre que dicho riesgo sea aceptable.
  - **Supervisar** continuamente los controles de seguridad del sistema de información, lo que incluye evaluar la eficacia de los controles, documentar los cambios en el sistema o su entorno operativo, realizar análisis del impacto de seguridad de los cambios asociados e informar sobre el estado de seguridad del sistema a los funcionarios designados de la organización.

Es importante destacar que el NIST también proporciona un conjunto de controles de seguridad para elegir, lo que simplifica la evaluación de riesgos. Recientemente, el Marco de Gestión de Riesgos (RMF) se ha modificado para incluir las cuestiones de privacidad.




## 4.7 Modelado de amenazas, priorización, y mitigación

Un método de modelado de amenazas (TMM) es un enfoque para crear una abstracción de un sistema de software, cuyo objetivo es identificar las capacidades y objetivos de los atacantes, y utilizar dicha abstracción para generar y catalogar las posibles amenazas que el sistema debe mitigar [Shu16].

STRIDE (acrónimo de seis categorías de amenazas) representa varios métodos de modelado de amenazas [Mea18] y es el TMM más consolidado, representando el estado actual de la práctica. En esencia, STRIDE requiere descomponer un sistema en sus diversos elementos, evaluar la vulnerabilidad de cada uno de ellos a las amenazas y, posteriormente, mitigarlas [Her06].





En la práctica, una implementación típica de STRIDE incluye modelar un sistema con diagramas de flujo de datos (DFD),<sup>5</sup> asignar los elementos del DFD a las seis categorías de amenazas, determinar las amenazas específicas mediante listas de verificación o árboles de amenazas y documentar las amenazas y los pasos para su prevención [Sca15].


STRIDE puede implementarse manualmente; Sin embargo, también se puede utilizar la herramienta gratuita Microsoft Secure Development Lifecycle (SDL) (Amenaza) Threat Modeling Tool [Mic17]. La Tabla 18.2 identifica la propiedad de seguridad asociada a cada una de las seis categorías de amenazas.

5 Se puede descargar un breve tutorial sobre diagramas de flujo de datos desde [https://ratandon.mysite.syr.edu/cis453/notes/DFD over Flowcharts.pdf](https://ratandon.mysite.syr.edu/cis453/notes/DFD%20over%20Flowcharts.pdf)

**TABLE 18.2****Threat categories and security properties**

Threat	Security Property
Spoofing	Authentication
Tampering	Integrity
Repudiation	Nonrepudiation
Information disclosure	Confidentiality
Denial of service	Availability
Elevation of privilege	Authorization

Suplantación de identidad Autenticación, Manipulación Integridad, Repudio, No repudio, Divulgación de información, Confidencialidad, Denegación de servicio ,Disponibilidad, Elevación de privilegios, Autorización




Los DFD están diseñados para mostrar el funcionamiento de un sistema mediante símbolos estándar que representan gráficamente la interacción entre **almacenes de datos** (*p. ej., bases de datos, archivos, registros*), **procesos** (*p. ej., DLL, servicios web*), **flujos de datos** (*p. ej., llamadas a funciones, llamadas a procedimientos remotos*) y **entidades externas** (*p. ej., personas, otros sistemas*) [Sho14].

Una vez completados, cada uno de estos elementos del sistema puede asociarse a su vez con una o más categorías de amenazas relevantes, como se muestra en la Tabla 18.3.

  
**TABLE 18.3**

**Threat  
categories of  
DFD system  
elements**

Element	Spoofing	Tampering	Repudiation	Information Disclosure	Denial of Service	Elevation of Privilege
Data flows		X		X	X	
Data stores		X		X	X	
Processes	X	X	X	X	X	X
External entity	X		X			



En la siguiente etapa, el usuario típico de STRIDE trabaja con una lista de verificación (*que puede tener la forma de un árbol de amenazas*) de amenazas específicas asociadas con cada coincidencia entre un elemento del DFD y una categoría de amenaza. Estas listas de verificación están disponibles a través de los libros o herramientas de referencia de STRIDE.

*Una vez identificadas las amenazas, se pueden desarrollar y priorizar estrategias de mitigación.*

*Normalmente, la priorización se basa en consideraciones de costo y valor.* Considerar el costo de implementar la estrategia de mitigación es importante, pero es igualmente importante considerar el costo de no implementarla, lo cual se refleja en el valor.

*Recuerde que los riesgos que se materializan resultan en costos que no solo se expresan en términos de dólares, sino que también pueden reflejar pérdida de reputación, pérdida de confianza e incluso la pérdida de vidas.*

## 4.8 Superficie de ataque

Una superficie de ataque se puede definir <sup>6</sup> de la siguiente manera:

La superficie de ataque describe todos los puntos por los que un atacante podría acceder a un sistema y por dónde podría extraer datos. La superficie de ataque de una aplicación es:

1. la suma de todas las rutas de entrada y salida de datos/comandos de la aplicación;
2. el código que protege estas rutas (incluyendo la conexión y autenticación de recursos, la autorización, el registro de actividad, la validación y codificación de datos);
3. todos los datos valiosos utilizados en la aplicación, incluyendo secretos y claves, propiedad intelectual, datos empresariales críticos, datos personales e información de identificación personal (PII); y
4. el código que protege estos datos (incluyendo cifrado y sumas de comprobación, auditoría de acceso y controles de integridad de datos y seguridad operativa). [OWA18]

<sup>6</sup> [https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Attack\\_Surface\\_Analysis\\_Cheat\\_Sheet.md](https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Attack_Surface_Analysis_Cheat_Sheet.md)

# Laravel Cheat Sheet

## Introduction

This *Cheatsheet* intends to provide security tips to developers building Laravel applications. It aims to cover all common vulnerabilities and how to ensure that your Laravel applications are secure.

The Laravel Framework provides in-built security features and is meant to be secure by default. However, it also provides additional flexibility for complex use cases. This means that developers unfamiliar with the inner workings of Laravel may fall into the trap of using complex features in a way that is not secure. This guide is meant to educate developers to avoid common pitfalls and develop Laravel applications in a secure manner.

You may also refer the [Enlightn Security Documentation](#), which highlights common vulnerabilities and good practices on securing Laravel applications.

## The Basics

- Make sure your app is not in debug mode while in production. To turn off debug mode, set your `APP_DEBUG` environment variable to `false`:

```
APP_DEBUG=false
```



- Make sure your application key has been generated. Laravel applications use the app key for symmetric encryption and SHA256 hashes such as cookie encryption, signed URLs, password reset tokens and session data encryption. To generate the app key, you may run the `key:generate` Artisan command:

La Fundación OWASP [OWA18] afirma que el análisis de la superficie de ataque está:

*... dirigido a los desarrolladores para comprender y gestionar los riesgos de seguridad de las aplicaciones al diseñar y modificar una aplicación, así como a los especialistas en seguridad de aplicaciones al realizar una evaluación de riesgos de seguridad.*

*El enfoque aquí es proteger una aplicación de ataques externos; no considera los ataques a los usuarios u operadores del sistema (por ejemplo, inyección de malware, ataques de ingeniería social), y se presta menos atención a las amenazas internas, aunque los principios siguen siendo los mismos. Es probable que la superficie de ataque interna sea diferente a la externa, y algunos usuarios pueden tener amplio acceso.*

El análisis de la superficie de ataque consiste en identificar qué partes de un sistema deben revisarse y probarse para detectar vulnerabilidades de seguridad.

*El objetivo del análisis de la superficie de ataque es comprender las áreas de riesgo de una aplicación, informar a los desarrolladores y especialistas en seguridad sobre qué partes de la aplicación están expuestas a ataques, encontrar maneras de minimizarlos y observar cuándo y cómo cambia la superficie de ataque y qué significa esto desde una perspectiva de riesgo.*





## 4.9 Codificación segura

La codificación segura es justo lo que su nombre indica: codificar de forma que no se introduzcan vulnerabilidades como resultado de errores de codificación.

*No es sorprendente que la mayoría de las vulnerabilidades de software se deban a prácticas de codificación descuidadas y erróneas, muchas de las cuales pueden evitarse fácilmente.*

Por ejemplo, una condición conocida como desbordamiento de búfer resulta de uno de los errores de codificación más conocidos y comunes. OWASP<sup>7</sup> lo describe de la siguiente manera:

*Un desbordamiento de búfer se produce cuando un programa intenta introducir más datos en un búfer de los que puede almacenar o cuando intenta introducir datos en un área de memoria más allá del búfer. En este caso, un búfer es una sección secuencial de memoria asignada para contener cualquier cosa, desde una cadena de caracteres hasta una matriz de enteros. Escribir fuera de los límites de un bloque de memoria asignada puede corromper datos, bloquear el programa o provocar la ejecución de código malicioso.*

El desbordamiento de búfer es solo un ejemplo de errores de codificación que pueden generar vulnerabilidades. Afortunadamente, existen varios estándares de codificación que ofrecen orientación sobre codificación segura.


El sitio web de SEI/CERT<sup>8</sup> ofrece una lista de las 10 mejores prácticas de codificación segura:

<sup>7</sup> See [https://www.owasp.org/index.php/Buffer\\_overflow\\_attack](https://www.owasp.org/index.php/Buffer_overflow_attack)

<sup>8</sup> See <https://wiki.sei.cmu.edu/confluence/display/seccode/Top+10+Secure+Coding+Practices>

**1. Validar la entrada.** Validar la entrada de todas las fuentes de datos no confiables.

**2. Prestar atención a las advertencias del compilador.** Compilar código utilizando el nivel de advertencia más alto disponible para el compilador y eliminar las advertencias modificando el código.



**3. Diseñar y diseñar políticas de seguridad.** Crear una arquitectura de software y diseñar el software para implementar y aplicar políticas de seguridad.

**4. Mantener la simplicidad.** Mantener el diseño lo más simple y compacto posible.

**5. Denegación predeterminada.** Basar las decisiones de acceso en el permiso en lugar de la exclusión.

**6. Adherirse al principio del mínimo privilegio.** Cada proceso debe ejecutarse con el mínimo conjunto de privilegios necesario para completar la tarea.

**7. Limpiar los datos enviados a otros sistemas.** Limpiar todos los datos que se pasan a subsistemas complejos, como shells de comandos, bases de datos relacionales y componentes comerciales listos para usar (COTS).

**8. Practicar la defensa a fondo.** Gestionar el riesgo con múltiples estrategias defensivas.

**9. Utilizar técnicas eficaces de control de calidad.**

**10. Adoptar un estándar de codificación segura.**



SEI CERT y otras organizaciones también proporcionan estándares de codificación segura.<sup>9</sup>

Además de utilizar un estándar de codificación segura, debe inspeccionar los errores de codificación que generen vulnerabilidades. Esto es simplemente un complemento natural a su proceso habitual de inspección y revisión de código (Capítulo 2). Las herramientas de análisis estático <sup>10</sup> se utilizan para analizar el código automáticamente y constituyen otro mecanismo para detectar vulnerabilidades causadas por errores de codificación.

<sup>9</sup> See <https://wiki.sei.cmu.edu/confluence/display/seccode/SEI+CERT+Coding+Standards>

<sup>10</sup> A list of commercially available tools can be found at [https://en.wikipedia.org/wiki/List\\_of\\_tools\\_for\\_static\\_code\\_analysis](https://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis)



## 4.10 Medición

Desarrollar medidas adecuadas de seguridad de software es un problema complejo, sobre el cual existen diferentes puntos de vista. *Por un lado, se pueden analizar los procesos de desarrollo seguidos y evaluar si el software resultante es seguro. Por otro lado, se puede observar la incidencia de vulnerabilidades e intrusiones exitosas y medirlas para evaluar la seguridad del software. Sin embargo, ninguno de estos enfoques de medición permite afirmar con total certeza que nuestro software es seguro.*

Al añadir software de soporte, como sistemas operativos y sistemas externos interoperables, la medición de la seguridad del software se vuelve aún más difícil. No obstante, se han logrado algunos avances.

*Las medidas de calidad del software pueden ser muy útiles para medir la seguridad del software.* En concreto, las vulnerabilidades invariablemente indican defectos de software. Aunque *no todos los defectos de software son problemas de seguridad, las vulnerabilidades en el software generalmente resultan de algún tipo de defecto, ya sea en los requisitos, la arquitectura o el código.*


Por lo tanto, medidas como el recuento de defectos y vulnerabilidades [Woo14] son útiles.

Microsoft utiliza medidas como el análisis de la superficie de ataque e intenta minimizar la superficie de ataque (lugares donde el software puede verse comprometido).

Así como el uso de modelos de madurez como **CMMI** sugiere que se obtendrá un software de mayor calidad, los procesos de desarrollo de seguridad maduros, como los que destaca **BSIMM** <sup>11</sup>, darán como resultado un software más seguro. En algunos casos, se anima a las organizaciones a identificar el conjunto único de métricas de seguridad que les son relevantes. **BSIMM** hace referencia a esto, al igual que **SAMM** <sup>12</sup>.

11 See <https://www.bsimm.com/>

12 See [https://www.owasp.org/index.php/OWASP SAMM Project](https://www.owasp.org/index.php/OWASP_SAMM_Project)



Es importante tener en cuenta que ninguna de las características de medición que implican los diversos modelos de madurez es perfecta.

Si se siguen buenos procesos de desarrollo de software seguro, ¿se garantiza que el software sea seguro? ¡No! Si se encuentran muchas vulnerabilidades, ¿significa que la mayoría ya se han encontrado o que aún quedan más por descubrir porque se trata de un software particularmente deficiente? No hay respuestas sencillas.

*Sin embargo, para evaluar las vulnerabilidades y la seguridad del software asociada, es necesario recopilar datos para poder analizar patrones a lo largo del tiempo. Si no recopilamos datos sobre la seguridad del software, nunca podremos medir su mejora.*

Las Tablas 18.4 y 18.5 ofrecen ejemplos de cómo evaluar la seguridad del software durante cada fase del ciclo de vida. Las tablas completas y el análisis se pueden encontrar en [Mea17] y [Alb10].

**TABLE 18.4****Examples of  
life-cycle-  
phase  
measures****Life-Cycle Phase****Example Software Security Measures**

Requirements engineering

Percentage of relevant software security principles reflected in requirements-specific actions (assuming security principles essential for a given development project have been selected)

Percentage of security requirements that have been subject to analysis (risk, feasibility, cost-benefit, performance trade-offs) prior to being included in the specification

Percentage of security requirements covered by attack patterns, misuse and abuse cases, and other specified means of threat modeling and analysis

Architecture and design

Percentage of architectural and design components subject to attack surface analysis and measurement

Percentage of architectural and design components subject to architectural risk analysis

Percentage of high-value security controls covered by a security design pattern



Ejemplo de fase del ciclo de vida: Medidas de seguridad del software

**Ingeniería de requisitos:** Porcentaje de principios de seguridad del software relevantes reflejados en acciones específicas de los requisitos (suponiendo que se han seleccionado los principios de seguridad esenciales para un proyecto de desarrollo determinado).

Porcentaje de requisitos de seguridad que se han analizado (riesgo, viabilidad, coste-beneficio, compensaciones de rendimiento) antes de su inclusión en la especificación.

Porcentaje de requisitos de seguridad cubiertos por patrones de ataque, casos de uso indebido y abuso, y otros medios específicos de modelado y análisis de amenazas.

**Arquitectura y diseño:** Porcentaje de componentes arquitectónicos y de diseño sujetos a análisis y medición de la superficie de ataque.

Porcentaje de componentes arquitectónicos y de diseño sujetos a análisis de riesgos arquitectónicos.

Porcentaje de controles de seguridad de alto valor cubiertos por un patrón de diseño de seguridad.

**TABLE 18.5****Example  
measures  
based on  
the seven  
principles  
of evidence**

Principle	Description
Risk	<p>Number of active and latent threats, categorized</p> <p>Incidents reported by category of threat</p> <p>Likelihood of occurrence for each threat category</p> <p>Financial and/or human safety estimate of impact for each threat category</p>
Trusted dependencies	<p>Number of levels of subcontracting in the supply chain (in other words, have the subcontractors, in turn, executed subcontracts, and what is the depth of this activity?)</p> <p>Number of suppliers by level</p> <p>Hierarchical and peer dependencies between suppliers by level</p> <p>Number of (vetted) trusted suppliers in the supply chain by level</p>

Descripción del principio

**Riesgo** Número de amenazas activas y latentes, categorizadas

Incidentes reportados por categoría de amenaza

Probabilidad de ocurrencia para cada categoría de amenaza

Estimación del impacto financiero o de seguridad humana para cada categoría de amenaza

**Dependencias de confianza** Número de niveles de subcontratación en la cadena de suministro (en otras palabras, ¿los subcontratistas, a su vez, han ejecutado subcontratos? ¿Y cuál es la profundidad de esta actividad?)

Número de proveedores por nivel

Dependencias jerárquicas y entre pares entre proveedores por nivel

Número de proveedores de confianza (verificados) en la cadena de suministro por nivel

## 4.11 Mejora del proceso de seguridad y modelos de madurez


Existen diversos modelos de mejora de procesos y madurez para el desarrollo de software en general, como el Modelo de Integración de Madurez de Capacidades (CMMI).<sup>13</sup>

Para la madurez de la ciberseguridad, el Instituto CMMI ofrece un producto más reciente: la plataforma de Gestión de Madurez de Capacidades Cibernéticas.<sup>14</sup> OWASP ofrece el Modelo de Madurez de Aseguramiento de Software (SAMM).<sup>15</sup> SAMM es un marco abierto que ayuda a las organizaciones a formular e implementar una estrategia de seguridad de software adaptada a los riesgos específicos que enfrenta la organización.

<sup>13</sup> See <https://cmmiinstitute.com/>


<sup>14</sup> See <https://cmmiinstitute.com/products/cybermaturity>

<sup>15</sup> See [https://www.owasp.org/index.php/OWASP\\_SAMM\\_Project](https://www.owasp.org/index.php/OWASP_SAMM_Project)



Un análisis exhaustivo de estos modelos queda fuera del alcance de este libro. Para ofrecer una visión general, considere el objetivo general de **SAMM**:

- Evaluar las prácticas de seguridad de software existentes en una organización.
- Desarrollar un programa de garantía de seguridad de software equilibrado en iteraciones bien definidas.
- Demostrar mejoras concretas en un programa de garantía de seguridad.
- Definir y medir las actividades relacionadas con la seguridad en toda la organización.



Quizás el modelo de madurez más conocido, específico para la seguridad del software, sea el Modelo de Madurez para la Construcción de la Seguridad (BSIMM). El BSIMM se publica periódicamente, generalmente cada uno o dos años. El modelo BSIMM y sus resultados resumidos de evaluación recientes se pueden descargar del sitio web de BSIMM.<sup>16</sup> Según los desarrolladores de BSIMM, está diseñado para ser utilizado por quienes crean y ejecutan iniciativas de seguridad del software.

***Todos los modelos de madurez mencionados aquí (y otros) tienen beneficios, y sus elementos esenciales están disponibles gratuitamente. Sin embargo, en ocasiones la evaluación la realizan entidades externas. Es posible realizar una autoevaluación interna y definir el programa de mejora correspondiente, pero requiere recursos y esfuerzo dedicados.***

Como alternativa, algunas de estas organizaciones ofrecen programas de evaluación, lo que proporciona una visión externa de las fortalezas y las áreas de mejora dentro de una organización de software.

<sup>16</sup> See <https://www.bsimm.com/>



## 4.12 Resumen

- Todos los ingenieros de software deben comprender lo que implica desarrollar software seguro. Los pasos necesarios para mejorar la seguridad de sus productos de software son relevantes en cada actividad del proceso de software, independientemente del modelo de proceso utilizado.
- Si bien aún existen muchas preguntas abiertas y tecnologías que requieren mayor investigación, hoy en día existen numerosos recursos disponibles para ayudar con este desafío. En cada actividad que normalmente se realiza en el proceso de software, procure incorporar aspectos de seguridad. Modelos como Microsoft SDL y el modelo SQUARE pueden evaluarse para determinar qué pasos puede incorporar en su proceso de desarrollo.

- Incorpore seguridad a una actividad de análisis de riesgos, especialmente utilizando la guía detallada disponible del NIST. Dada la cantidad de estándares de codificación segura disponibles, cualquiera puede aprender a codificar de forma segura. Inspeccione su código para detectar vulnerabilidades restantes.
- Aprenda a identificar vulnerabilidades de seguridad y a desarrollar y priorizar estrategias de mitigación. Realice pruebas de análisis estático en su código. Visite los sitios web de OWASP y BSIMM, entre otros, para obtener información sobre la madurez en la ingeniería de seguridad del software.
- A medida que el software se vuelve cada vez más omnipresente, también crece el número de vulnerabilidades y ataques informáticos exitosos. Requeriremos todos nuestros esfuerzos para frenar esta tendencia, pero muchas de las herramientas ya existen para abordar este problema. Las consecuencias de no abordar la seguridad del software son graves, y los beneficios de desarrollar software seguro son enormes.





# BIBLIOGRAFÍA

*Ingeniería del software UN ENFOQUE PRÁCTICO 9na EDICIÓN*

*Roger S. Pressman, Ph.D.*

*University of Connecticut*