Pruebas de software

La presente guía está basada en **OWASP Testing Project**.

El **objetivo** de dicho proyecto es crear un marco de trabajo para ayudar a la gente a entender el *qué*, *por qué*, *cuándo*, *dónde* y *cómo* de las pruebas de aplicaciones web. Este marco ayuda a las organizaciones a probar sus aplicaciones web con el fin de crear software confiable y seguro.

TODO - acabarlo al final de reescribir

El resto de la presente guía está organizada de la siguiente manera: ....

Medición de la seguridad: la economía del software inseguro

Un principio básico de la ingeniería de software se resume en una cita de Controlling Software Projects: Management, Measurement, and Estimates por Tom DeMarco:

> No puedes controlar lo que no puedes medir.

Un aspecto que debe destacarse es que las medidas de seguridad se refieren tanto a **cuestiones técnicas concretas** (por ejemplo, la prevalencia de una determinada vulnerabilidad) como a la forma en que esas **cuestiones afectan a la economía del software**. La mayoría de los técnicos comprenderán al menos los problemas básicos, o tal vez tengan una comprensión más profunda de las vulnerabilidades. Lamentablemente, **pocos son capaces de traducir esos conocimientos técnicos en términos monetarios** y cuantificar el costo potencial de las vulnerabilidades para el negocio del propietario de la aplicación. Hasta que esto no ocurra, los **CIOs** (Chief Information Officer) no podrán desarrollar un retorno preciso de la inversión (**ROI** - Return Of Investmet) en seguridad y, posteriormente, asignar presupuestos adecuados para la seguridad del software.

TODO Aquí falta la imagen que mostraba el costo de los errores en la cadena **shift left**

También es conocido que el costo de subsanar un error es mucho menor en las primeras fases del desarrollo de software, como muestra la siguiente figura, basada en el concepto de Shift Left Testing

En el antiguo modelo de desarrollo las pruebas sólo se realizaban en las últimas fases de la línea del ciclo de desarrollo:

Lo que producía un cuello de botella

Aplicando la práctica de encontrar y subsanar los errores en las primeras fase del desarrollo se mejora la calidad del mismo.

De ahí el nombre de esta práctica: **Shift Left**

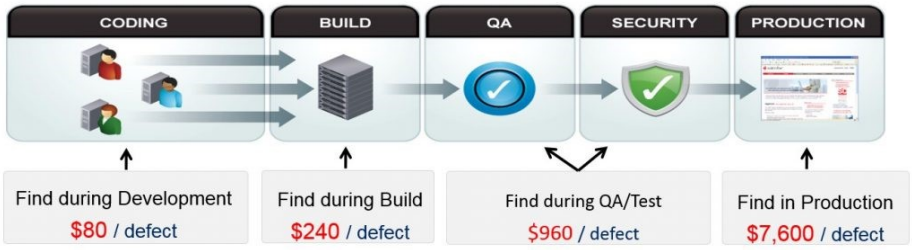

Figure 1: Shift Left

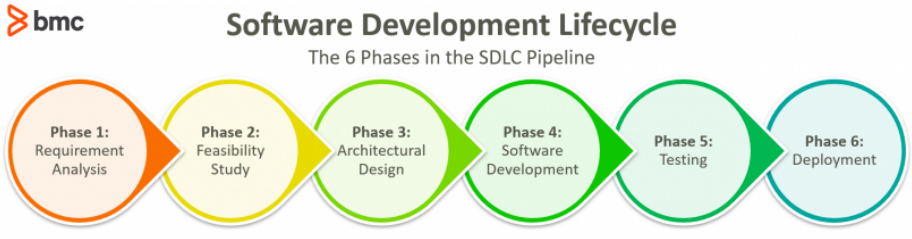

Figure 2: Software Development Lifecycle

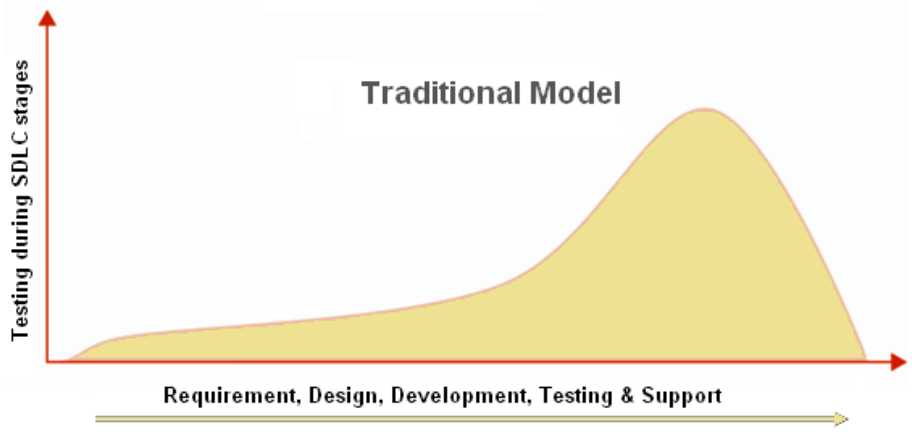

Figure 3: Modelo tradicionnal

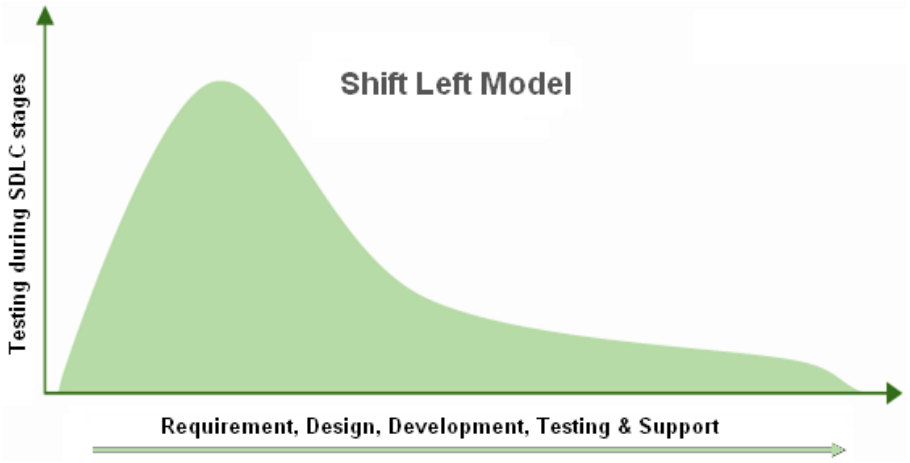

Figure 4: Shift Left Model

¿Qué son las pruebas (o tests)?

Muchas cosas necesitan ser probadas durante el ciclo de vida de desarrollo de una aplicación web, pero ¿qué significa realmente probar? El Diccionario de la Real Academia Española define "test" como:

> **test** (sustantivo): procedimiento destinado a establecer la calidad, el rendimiento o la fiabilidad de algo, especialmente antes de su uso generalizado.

Para los propósitos de este documento, la **prueba es un proceso de comparación del estado de un sistema o aplicación con un conjunto de criterios**. En la industria de la seguridad, las personas frecuentemente hacen pruebas contra un conjunto de criterios mentales que no están ni bien definidos ni completos. Como resultado de esto, muchos neófitos consideran las pruebas de seguridad como un arte oscuro.

¿Por qué realizar las pruebas?

Como ya se ha comentado anteriormente las pruebas son necesarias, entre otras cosas, porque disminuyen el coste de desarrollo de software.

¿Cuándo probar?

Cuanto antes mejor.

La mayoría de la gente hoy en día no prueba el software hasta que ya ha sido creado y está en la fase de despliegue de su ciclo de vida (es decir, el código ha sido creado e instanciado en una aplicación web que funciona). Esta es generalmente una práctica muy ineficaz y de costo prohibitivo. Uno de los mejores métodos para evitar que aparezcan errores de seguridad en las aplicaciones de producción es mejorar el ciclo de vida del desarrollo de software (Software Development Life Cycle SDLC) incluyendo la seguridad en cada una de sus fases. Un SDLC es una estructura impuesta en el desarrollo de artefactos de software. La siguiente figura muestra un modelo SDLC genérico, así como el coste creciente (estimado) de arreglar los fallos de seguridad en dicho modelo.

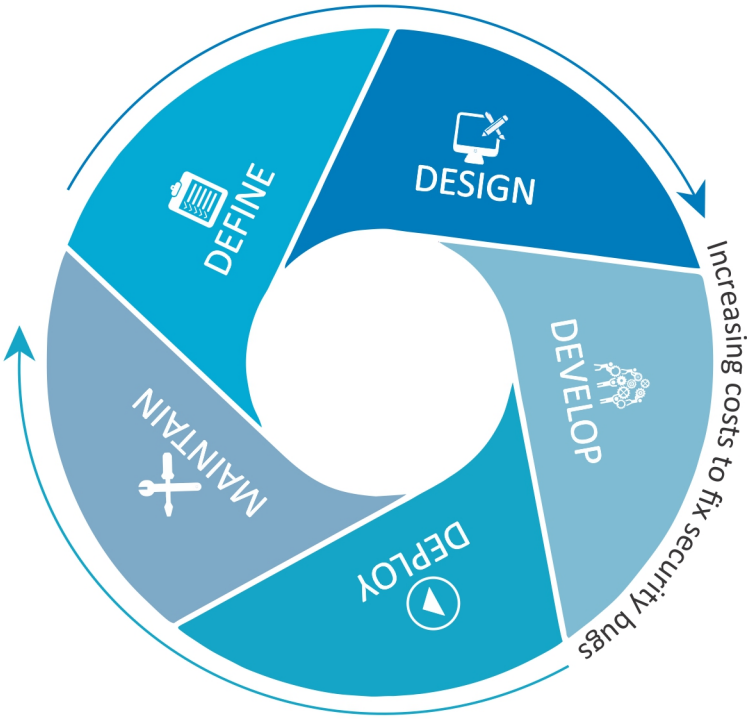

Figure 5: Generic SDLC Model

¿Qué hay que probar?

Puede ser útil pensar en el desarrollo de software como una combinación de personas, procesos y tecnología. Si estos son los factores que "crean" el software, entonces es lógico que estos sean los factores que deben ser probados. Hoy en día, la mayoría de la gente generalmente prueba la tecnología o el propio software.

Un programa de pruebas eficaz debe tener componentes que comprueben lo siguiente:

Personas - para asegurar que haya una educación y concienciación adecuadas;

Proceso - para asegurar que haya políticas y normas adecuadas y que la gente sepa cómo seguir esas políticas;

Tecnología - para asegurar que el proceso ha sido efectivo en su implementación.

Principios de la prueba

Hay algunos conceptos erróneos comunes cuando se desarrolla una metodología de pruebas para encontrar errores de seguridad en el software. Este capítulo cubre algunos de los principios básicos que los profesionales deben tener en cuenta cuando realizan pruebas de seguridad en el software.

No hay ninguna bala de plata

Aunque es tentador pensar que un escáner de seguridad o un cortafuegos de aplicación (WAF) proporcionará muchas defensas contra los ataques o identificará una multitud de problemas, en realidad no hay una bala de plata para el problema del software inseguro. El software de evaluación de la seguridad de las aplicaciones, si bien es útil como primera pasada para encontrar frutos de poca monta, suele ser inmaduro e ineficaz en las evaluaciones a fondo o para proporcionar una cobertura de pruebas adecuada. Recuerda que la seguridad es un proceso y no un producto.

Pensar estratégicamente, no tácticamente

Los profesionales de la seguridad se han dado cuenta de la falacia del modelo de parchear y penetrar que fue omnipresente en la seguridad de la información durante la década de 1990. El modelo de parchear y penetrar **implica arreglar un error reportado, pero sin la investigación adecuada de la causa de fondo**. Este modelo suele asociarse con la **ventana de vulnerabilidad**, también denominada **ventana de exposición**, que se muestra en la figura siguiente. La evolución de las vulnerabilidades en los programas informáticos

comunes utilizados en todo el mundo ha demostrado la ineficacia de este modelo. Para más información sobre las ventanas de exposición, véase Schneier sobre Seguridad.

Estudios de vulnerabilidad demuestran que con el tiempo de reacción de los atacantes en todo el mundo, la típica ventana de vulnerabilidad no proporciona suficiente tiempo para la instalación de parches, ya que el tiempo que transcurre entre que se descubre una vulnerabilidad y se desarrolla y publica un ataque automatizado contra ella disminuye cada año. Es decir, los atacantes son más rápidos en descubrir vulnerabilidades que las empresas en solventarlas.

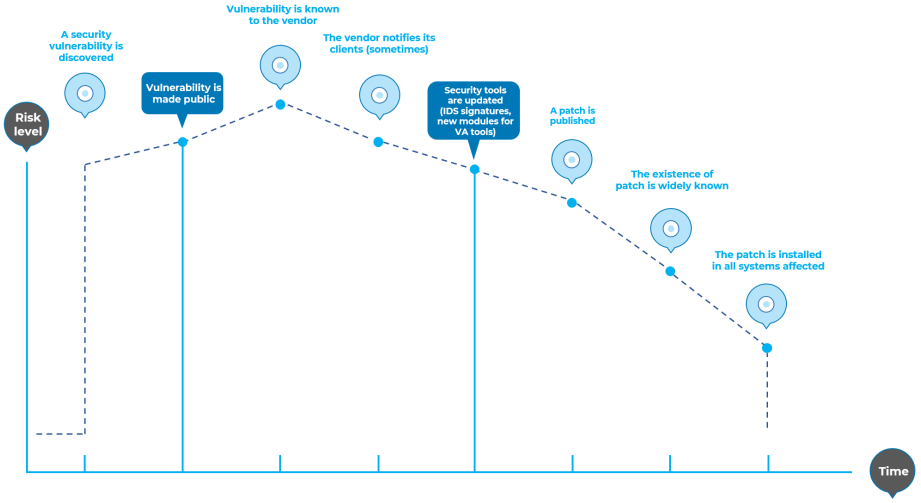

Figure 6: Window of Vulnerability

Existen varios marcos SDLC seguros como son el Modelo de Madurez de Garantía de Software Abierto (OpenSAMM) de la OWASP y la ISO/IEC 27034. Si alguien desea profundizar en estos temas puede consultar la información referenciada, ya que su explicación queda fuera del ámbito de este documento.

Prueba de forma temprana y frecuentemente

Cuando un error se detecta a tiempo en el SDLC puede ser abordado más rápido y a un menor costo. Un paso clave para que esto sea posible es educar a los equipos de desarrollo y de control de calidad sobre los problemas de seguridad comunes y las formas de detectarlos y prevenirlos. Aunque las nuevas bibliotecas, herramientas o lenguajes pueden ayudar a diseñar programas con menos errores de seguridad, constantemente surgen nuevas amenazas y los desarrolladores deben ser conscientes de las amenazas que afectan al software que están desarrollando. La educación en pruebas de seguridad también ayuda a los desarrolladores a adquirir la mentalidad adecuada para probar una aplicación desde

la perspectiva de un atacante. Esto permite a cada organización considerar los problemas de seguridad como parte de sus responsabilidades actuales.

Automatización de pruebas

En metodologías de desarrollo modernas como (pero no limitado a): agile, devops / devsecops o desarrollo rápido de aplicaciones (RAD), se debe considerar la integración de pruebas de seguridad en los flujos de trabajo de Integración continua/Implementación continua (CI/CD - Continuos Integration/Continuos Development) para mantener la información / análisis de seguridad de referencia e identificar las debilidades del tipo "**low-hanging fruits**".

> En la comunidad de seguridad de la información, **low-hanging fruits** son las técnicas más fáciles y frecuentes que un atacante puede utilizar para obtener acceso a los sistemas y datos

Comprender el alcance de la seguridad

Es importante saber cuánta seguridad requerirá un proyecto determinado. Los bienes que se van a proteger deben recibir una clasificación que indique cómo se van a manejar. No todos los proyectos necesitan tener el mismo nivel de confianza. Pensemos, por ejemplo, un software destinado a una intranet no necesita de la misma seguridad como un software médico con historiales de pacientes al que se pueda acceder mediante Internet o con una aplicación móvil.

Como modelo nos puede ayudar el Open Web Application Security Project (versión en PDF) de tal forma que podamos cumplir el Esquema Nacional de Seguridad (**ENS**). Y no nos hemos de olvidar de la protección de datos de carácter personal.

Desarrollar la mentalidad correcta

Probar con éxito una aplicación para las vulnerabilidades de seguridad requiere pensar "outside the box". Los **casos de uso normales** probarán el comportamiento normal de la aplicación cuando un usuario la está usando de la manera que se espera. **Una buena prueba de seguridad requiere ir más allá de lo que se espera y pensar como un atacante que está tratando de romper la aplicación**. El **pensamiento creativo** puede ayudar a determinar qué datos inesperados pueden hacer que una aplicación falle de manera insegura. También puede ayudar a encontrar cualquier suposición hecha por los desarrolladores web que no siempre es cierta, y cómo esas suposiciones pueden ser subvertidas. Una de las razones por las que las herramientas automatizadas (cuyo funcionamiento recae en tareas de hacking ético) hacen un mal trabajo de comprobación de vulnerabilidades es que las herramientas automatizadas no

piensan de forma creativa. El pensamiento creativo debe hacerse caso por caso, ya que la mayoría de las aplicaciones web se están desarrollando de una manera única (incluso cuando se utilizan marcos de trabajo comunes).

Comprender el sujeto de la aplicación

Una de las primeras iniciativas importantes de todo buen programa de seguridad debería ser exigir **la documentación exacta de la aplicación**. La arquitectura, los diagramas de flujo de datos, los casos de uso, etc., deben registrarse en documentos oficiales y ponerse a disposición para su examen. Los documentos de especificación técnica y de aplicación deben incluir información que enumere no sólo los **casos de uso deseados, sino también cualquier caso de uso específicamente desautorizado**. Por último, es bueno contar por lo menos con una infraestructura de seguridad básica que permita la vigilancia y el seguimiento de los ataques contra las aplicaciones y la red de una organización (por ejemplo, los sistemas IDS).

Usar las herramientas adecuadas

Si bien ya hemos dicho que no existe una herramienta de bala de plata, las herramientas juegan un papel fundamental en el programa de seguridad general. Hay una gama de herramientas comerciales y de código abierto que pueden automatizar muchas tareas de seguridad rutinarias. Estas herramientas pueden simplificar y acelerar el proceso de seguridad ayudando al personal de seguridad en sus tareas. Sin embargo, es importante entender exactamente lo que estas herramientas pueden y no pueden hacer para que no se vendan en exceso o se utilicen incorrectamente.

El Diablo está en los detalles

Es fundamental no realizar un examen de seguridad superficial de una solicitud y considerarla completa. Esto infundirá una falsa sensación de confianza que puede ser tan peligrosa como no haber hecho una revisión de seguridad en primer lugar. Es vital revisar cuidadosamente los hallazgos y eliminar cualquier falso positivo que pueda quedar en el informe. Informar de un hallazgo de seguridad incorrecto puede a menudo socavar el mensaje válido del resto del informe de seguridad. Se debe tener cuidado en verificar que cada posible sección de la lógica de la aplicación ha sido probada, y que cada escenario de caso de uso fue explorado para posibles vulnerabilidades.

Usar el código fuente cuando esté disponible

Si bien los resultados de las pruebas de penetración de la caja negra (**black-box pentesting**) pueden ser impresionantes y útiles para demostrar la forma

en que se exponen las vulnerabilidades en un entorno de producción, no son la forma más eficaz o eficiente de asegurar una aplicación. Es difícil para las pruebas dinámicas probar toda la base de código, en particular si existen muchas declaraciones condicionales anidadas. Si el código fuente de la aplicación está disponible, debería entregarse al personal de seguridad para ayudarles en su revisión. Es posible descubrir vulnerabilidades dentro de la fuente de la aplicación que se pasarían por alto durante una intervención de caja negra.

Desarrollar métricas

Una parte importante de un buen programa de seguridad es la capacidad de determinar si las cosas están mejorando. Es importante rastrear los resultados de los compromisos de prueba, y desarrollar métricas que revelen las tendencias de seguridad de la aplicación dentro de la organización.

Las buenas métricas lo demostrarán:

- Si se requiere más capacitación y formación;
- Si hay un mecanismo de seguridad particular que no es claramente comprendido por el equipo de desarrollo;
- Si el número total de problemas relacionados con la seguridad que se encuentran cada mes disminuye.

Las métricas coherentes que pueden generarse de forma automatizada a partir del código fuente disponible también ayudarán a la organización a evaluar la eficacia de los mecanismos introducidos para reducir los errores de seguridad en el desarrollo de programas informáticos. Las métricas no son fáciles de desarrollar, por lo que el uso de métricas estándar como las proporcionadas por el proyecto OWASP Metrics y otras organizaciones es un buen punto de partida.

Documentar los resultados de la prueba

Para concluir el proceso de pruebas, es importante producir un registro formal de las medidas de prueba que se tomaron, por quién, cuándo se realizaron y los detalles de los resultados de las pruebas. Es conveniente acordar un formato aceptable para el informe que sea útil para todas las partes interesadas, entre las que pueden figurar los promotores, la gestión de proyectos, los propietarios de empresas, el departamento de tecnología de la información, la auditoría y el cumplimiento.

En el informe se debe identificar claramente al propietario del negocio **dónde existen riesgos materiales**, y hacerlo de manera suficiente para obtener su respaldo para las m**edidas de mitigación** subsiguientes. El informe también debe ser claro para el promotor al señalar la f**unción exacta que se ve afectada por la vulnerabilidad** y las **recomendaciones conexas** para resolver

los problemas en un lenguaje que el promotor comprenda. El informe también **debe permitir que otro probador de seguridad reproduzca los resultados**. Escribir el informe no debería ser una carga excesiva para los propios probadores de seguridad. Los probadores de seguridad no son generalmente reconocidos por sus habilidades de escritura creativa, y acordar un informe complejo puede llevar a instancias en las que los resultados de la prueba no estén documentados adecuadamente. El uso de una plantilla de informe de prueba de seguridad puede ahorrar tiempo y asegurar que los resultados se documenten de forma precisa y consistente, y que estén en un formato adecuado para la audiencia.

Inspecciones y revisiones manuales

Las inspecciones manuales son revisiones humanas que normalmente prueban las implicaciones de seguridad de **las personas, los procesos y las tecnologías**. Normalmente se llevan a cabo **analizando la documentación** o realizando entrevistas con los diseñadores o los propietarios de los sistemas.

Sabiendo cómo y por qué se ha desarrollado algo es probable que se pueda inferir si existe alguna preocupación relacionada con la seguridad.

Ventajas

- No requiere ninguna tecnología de apoyo
- Puede aplicarse a una variedad de situaciones
- Flexible
- Promueve el trabajo en equipo
- Se realiza al principio del SDLC

Desventajas

- Puede llevar mucho tiempo
- El material de apoyo no siempre está disponible
- Requiere un pensamiento y una habilidad humana significativa para ser efectivo

Modelización de amenazas

El modelado de amenazas es la práctica de identificar y priorizar amenazas potenciales y mitigaciones de seguridad para proteger algo de valor, como datos confidenciales o propiedad intelectual. Mediante el modelado continuo de las aplicaciones de amenazas, los equipos de seguridad pueden proteger mejor las aplicaciones al tiempo que educan al equipo de desarrollo y crean una cultura de seguridad en toda la empresa. Es un buen comienzo para empezar una

cultura de devsecops en la empresa. Se puede consultar más información en https://en.wikipedia.org/wiki/Threat_model

En el caso del desarrollo de software se puede seguir la OWASP Code Review Project (versión en PDF)

Ventajas

- La visión práctica del atacante del sistema
- Flexible
- Al principio del SDLC

Desventajas

- Una técnica relativamente nueva
- Los buenos modelos de amenaza no significan automáticamente un buen software

Revisión del código fuente

La revisión del código fuente es el proceso de comprobar manualmente el código fuente de una aplicación web para detectar problemas de seguridad. Muchas vulnerabilidades graves de seguridad no pueden detectarse con ninguna otra forma de análisis o pruebas. Como dice el dicho popular "si quieres saber lo que realmente está pasando, ve directamente a la fuente". Casi todos los expertos en seguridad están de acuerdo en que no hay sustituto para revisar realmente el código. Toda la información para identificar los problemas de seguridad está en el código, en algún lugar. A diferencia de las pruebas de software cerrado de terceros, como los sistemas operativos, cuando se prueban aplicaciones web (especialmente si han sido desarrolladas internamente) el código fuente debería estar disponible para fines de prueba.

Muchos problemas de seguridad no intencionados pero significativos son también extremadamente difíciles de descubrir con otras formas de análisis o pruebas, como las pruebas de penetración. Esto hace que el análisis del código fuente sea la técnica preferida para las pruebas técnicas. Con el código fuente, un probador puede determinar con precisión lo que está sucediendo (o se supone que está sucediendo) y eliminar las suposiciones de las pruebas de caja negra.

Ejemplos de cuestiones que son particularmente propicias para ser encontradas a través de las revisiones de código fuente incluyen problemas de concurrencia, lógica comercial defectuosa, problemas de control de acceso y debilidades criptográficas, así como puertas traseras, troyanos, huevos de pascua, bombas de tiempo, bombas lógicas y otras formas de código malicioso. Estos problemas se manifiestan a menudo como las vulnerabilidades más perjudiciales de los sitios web. El análisis del código fuente también puede ser sumamente eficiente para

encontrar problemas de aplicación, como lugares en los que no se ha realizado la validación de los datos o en los que pueden existir procedimientos de control abiertos a fallos. También es necesario revisar los procedimientos operacionales, ya que el código fuente que se está desplegando podría no ser el mismo que el que se está analizando aquí.

Hoy en día existen herramientas automatizadas para buscar vulnerabilidades en el código fuente. Un ejemplo de esto es CodeQL que descubre vulnerabilidades en código fuente a partir de una base de código. Permite consultar código como si fueran datos encontrando las variantes de una vulnerabilidad y erradicándolas para siempre.

Ventajas

- Completitud y eficacia
- Precisión
- Rápido (para revisores competentes)

Desventajas

- Requiere desarrolladores de seguridad altamente calificados
- Pueden faltar problemas en las bibliotecas compiladas
- No pueden detectar fácilmente los errores de tiempo de ejecución
- El código fuente realmente desplegado podría diferir del que se está analizando

Para más información sobre la revisión de códigos, vea el proyecto de revisión de códigos de OWASP.

Prueba de penetración

La prueba de penetración ha sido una técnica común utilizada para probar la seguridad de la red durante muchos años. También se conoce comúnmente como pruebas de caja negra o hacking ético. La prueba de penetración es esencialmente el "arte" de probar una aplicación en ejecución de forma remota para encontrar vulnerabilidades de seguridad, sin conocer el funcionamiento interno de la propia aplicación. Típicamente, el equipo de prueba de penetración es capaz de acceder a una aplicación como si fueran usuarios. El probador actúa como un atacante e intenta encontrar y explotar las vulnerabilidades. En muchos casos, el probador recibirá una cuenta válida en el sistema.

Si bien las pruebas de penetración han demostrado ser eficaces en la seguridad de las redes, la técnica no se traduce naturalmente en las aplicaciones. Cuando se realizan pruebas de penetración en redes y sistemas operativos, la mayor parte del trabajo consiste en encontrar, y luego explotar, las vulnerabilidades

conocidas en tecnologías específicas. Como las aplicaciones web están casi exclusivamente hechas a medida, las pruebas de penetración en el ámbito de las aplicaciones web se asemejan más a la investigación pura. Se han desarrollado algunas herramientas automatizadas de prueba de penetración, pero teniendo en cuenta la naturaleza a medida de las aplicaciones web, su eficacia por sí sola suele ser escasa.

Ventajas

- Puede ser rápido (y por lo tanto barato)
- Requiere un conjunto de habilidades relativamente más bajo que la revisión del código fuente
- Prueba el código que está siendo expuesto

Desventajas

- Demasiado tarde en el SDLC
- Sólo pruebas de impacto frontal

Referencias

OWASP Web Security Testing Guide

https://owasp.org/www-project-web-security-testing-guide/latest/

Shift Left

https://www.bmc.com/blogs/what-is-shift-left-shift-left-testing-explained/

ISO 27034 Seguridad de aplicaciones

https://blog.segu-info.com.ar/2020/03/isoiec-27034-seguridad-en-las.html

Protección de datos de carácter personal

https://www.boe.es/legislacion/codigos/codigo.php?id=55&modo=1&nota=0&tab=2

https://www.aepd.es/es/prensa-y-comunicacion/notas-de-prensa/aepd-publica-herramienta-comunica-brecha-rgpd

Métricas

https://www.guru99.com/software-testing-metrics-complete-tutorial.html

https://www.getzephyr.com/resources/whitepapers/qa-metrics-value-testing-metrics-within-software-development

https://owasp.org/www-project-security-qualitative-metrics/