Unidad 2 / Escenario 4

Lectura fundamental

Seguridad en las etapas de pruebas, despliegue y mantenimiento

Contenido

Palabras clave: pruebas, despliegue, mantenimiento, vulnerabilidades, pentesting, hardening, cambio, incidentes.

1. Introducción

Antes de iniciar, recapitulemos los temas visto en el escenario anterior, los cuales están hacen parte de la secuencia de temas revisados:

- Seguridad en la etapa de análisis:
  - Requerimientos: conceptos básicos y legales / contractuales
- Seguridad en la etapa de diseño:
  - Análisis de riesgos y modelamiento de amenazas
- Seguridad en la etapa de implementación:
  - Codificación segura
  - Revisión de código

En este escenario, se entenderá en principio como corroborar que estas actividades están siendo efectivas, para lo cual entenderemos 2 técnicas de pruebas de seguridad.

Posteriormente verificaremos que las acciones correctivas que haya sido necesario implementar por los hallazgos de las pruebas, hayan sido efectivos. De igual manera entenderemos e concepto de hardening o aseguramiento, con el cuál se asegura el ambiente donde se ejecuta la aplicación.

Finalmente y para cerrar el ciclo, aprenderemos 2 procesos que hacen parte de las operaciones de seguridad cuando la aplicación ya está en ambiente productivo.

Repasemos cuáles son las actividades de seguridad para las últimas 3 fases del ciclo de desarrollo de software según el modelo en cascada:

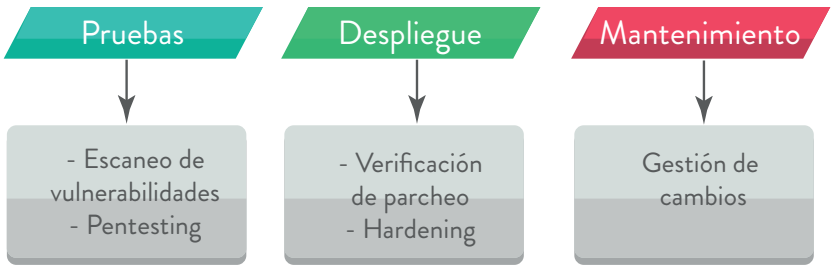

Figura 1. Actividades de seguridad para cada las últimas etapas del modelo en cascada
Fuente: elaboración propia

2. Seguridad en la etapa de pruebas

2.1. Escaneo de vulnerabilidades

Como lo vimos en el análisis de riesgos, el concepto de vulnerabilidad está definido como una debilidad de un activo o de control que puede ser explotado por una o más amenazas.

El escaneo de vulnerabilidades, que en realidad hace parte de la gestión de vulnerabilidades, es un proceso que tiene como objetivo identificar y tratar las debilidades de seguridad en las aplicaciones Web y en la infraestructura donde la aplicación está instalada.

Una representación sencilla del proceso de gestión de vulnerabilidades se observa en la siguiente imagen:

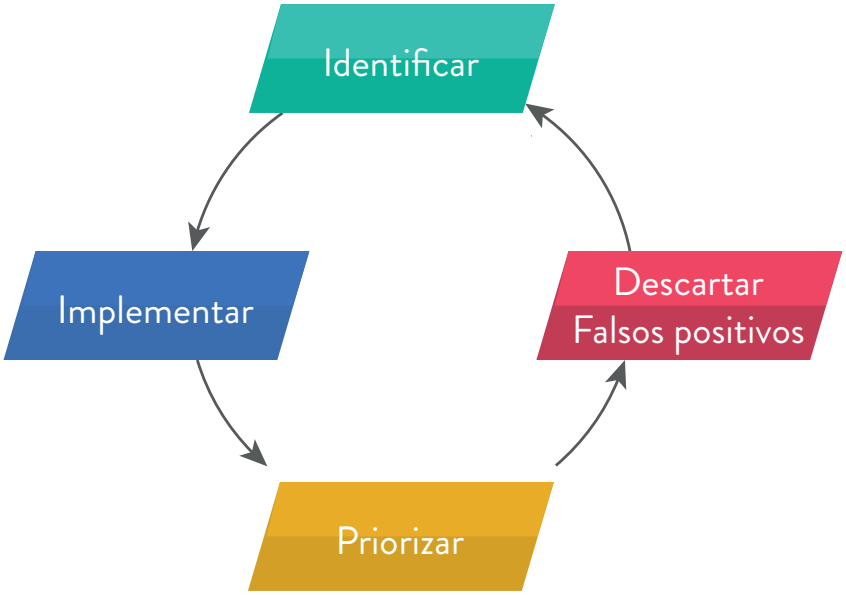

Figura 2. Proceso de gestión de vulnerabilidades
Fuente: elaboración propia

Como se puede concluir de la figura 1, la gestión de vulnerabilidades es un proceso continuo. No se existe un momento en el tiempo donde se deba interrumpir o dar por terminado el ejercicio, pues cada día aparecen nuevas vulnerabilidades que deben ser tratadas o remediadas. Veamos entonces a que corresponde cada una de las cajas:

» Identificar: La identificación de vulnerabilidades se puede realizar de las siguientes maneras:

- Manualmente: expertos en pruebas de vulnerabilidad, hacking ético y pentesting ejecutan comandos o realizan pruebas "manuales" mediante las cuales se identifican vulnerabilidades en las aplicaciones Web. La identificación manual es muy poco utilizada, siendo las herramientas la forma más utilizada para la ejecución del ejercicio.
- Herramientas: con el objetivo de automatizar el proceso, existen diferentes herramientas que ahorran grandes cantidades de tiempo en la identificación de vulnerabilidades. Las herramientas existentes para la identificación pueden ser:
  - Pagas: herramientas como Acunetix, Nexpose, Netsparquer o Nessus son ejemplos de algunas de ellas. Algunas de estas herramientas de pago ofrecen demos o trials para probar su funcionalidad sin ningún costo.
  - Gratuitas: OpenVAS, Wapiti, ZAP, Vega o distribuciones completas de hacking ético como Kali Linux se encuentran en esta categoría.
- El resultado de la identificación es un informe que incluye generalmente la siguiente información:
  - Código o identificador de la vulnerabilidad
  - Criticidad (cuantitativo o cualitativo, por ejemplo Crítica, Alta, Media, Baja, Informacional)
  - Código CVE
  - Recomendaciones para remediarla (solución técnica)

» Descartar falsos positivos: Los escáneres no son infalibles, es decir no ofrecen certeza del 100% de que lo reportado corresponde a la realidad pues en ocasiones se reportan vulnerabilidades inexistentes. Teniendo en cuenta esto, es necesario realizar un proceso de descarte de falsos positivos, para lo cual es necesario:

- Entender la vulnerabilidad: entendimiento técnico de la vulnerabilidad, es decir que software está vulnerable, cuál sería el impacto de no remediarlo y como podría ser explotada.
- Comprobarla: realizar verificaciones técnicas de la vulnerabilidad.
- Documentar: documentación del falso positivo, con lo cual se entenderá que no es necesario seguir trabajando en ella.

Ejemplo de un falso positivo:

- Se reportan vulnerabilidades para una versión de motor de base de datos en un servidor.
- El servidor no tiene ningún motor instalado.
- Se documenta o se marca la vulnerabilidad como falso positivo, con evidencias (reportes, pantallazos, archivos de configuración, etc.)

» Priorizar:

- Los escáneres califican las vulnerabilidades en niveles cuantitativos y cualitativos. Por ejemplo Nessus las clasifica así: Críticas, Altas, Medias, Bajas e Informacionales.
- A continuación se debe establecer un plan de tratamiento de vulnerabilidades que incluya al menos:
  - Identificación o código
  - Descripción
  - Remediación (lo más detallada posible)
  - Responsable de la remediación
  - Fecha de resolución
  - Fecha de seguimiento
- Finalmente se organizan de la más a la menos crítica, donde las fechas más cercanas o tempranas son asignadas a las más críticas.

» Implementar:

- Consiste en aplicar los cambios/modificaciones/configuraciones que cierran la vulnerabilidad.
- Posteriormente se repite el ejercicio, típicamente bajo las siguientes políticas:
  - Al menos una vez al año.
  - Cuando existe un cambio de versión del software o existe un cambio que realiza modificaciones en gran cantidad o importancia.
  - Cuando se cambian los controles de seguridad, por ejemplo cambio en el módulo o formulario de autenticación de una aplicación web.

¿Sabía qué...?

La gestión de vulnerabilidades es una de los controles más básicos e importantes en la gestión de la seguridad informática.

2.2. Pentesting

El pentesting o pruebas de penetración por su traducción tiene las siguientes características:

- Básicamente es un ejercicio que busca además de identificar vulnerabilidades, explotarlas para observar hasta donde se puede llegar.
- Existen diferentes tipos de pentest:
  - Según el conocimiento: pueden ser de caja negra, gris o blanca, es decir sin conocimiento, con conocimiento parcial (como la URL de la aplicación) o con conocimiento total (URL, direcciones IP, usuarios de prueba, manual de la aplicación, etc.).
  - Según la ubicación: las pruebas pueden ejecutarse desde una red externa o desde una red interna. Generalmente los resultados varian, pues entra en juego otro tipo de controles, por ejemplo la seguridad perimetral (IPS/IDS, WAF, balanceadores de carga, etc.).
- Las etapas básicas de un pentesting son:
  - Recolectar información
  - Escanear
  - Explotar
  - Remediar
  - Reportar

- Profundicemos en la etapa de “Explotar”:
  - Una vulnerabilidad identificada puede tener exploits (líneas de código que explotan la vulnerabilidad) ya desarrollados o se pueden desarrollar. Los exploits ya desarrollados se encuentran en bases de datos como Exploit-DB, los exploits a desarrollar generalmente se hacen en el lenguaje Python.
  - Una vez se explota la vulnerabilidad, se busca ganar o elevar privilegios. Un ejemplo de esto es lograr acceso a una aplicación web con un usuario restringido, la elevación de privilegios consistiría entonces en intentar obtener los privilegios de un usuario administrador.
  - Se toman evidencias de hasta donde se pudo llegar, con las correspondientes imágenes o reportes.

3. Seguridad en la etapa de despliegue

3.1. Verificación de parcheo

Una vez se han terminado de remediar las vulnerabilidades de acuerdo al plan de tratamiento o remediación, se debe verificar su cierre. Para realizar esto, es necesario:

- Re-ejecutar las herramientas de escaneo de vulnerabilidades. Algunas herramientas traen la funcionalidad de generar reportes diferenciales, lo cual facilita la comprobación del cierre.
- Lanzar nuevamente los ataques que fueron satisfactorios, para verificar que ya no sea posible.

Cuando se verifica el parcheo sobre una vulnerabilidad, pueden obtenerse 3 posibles resultados:

- La vulnerabilidad está remediada, en cuyo caso se documenta con evidencias y se cierra la vulnerabilidad.
- La vulnerabilidad permanece abierta, en cuyo caso se deberá reportar la situación al responsable del cierre para que tome las medidas correctivas a que haya lugar. De igual manera se recomienda:
  - Verificar si se implementó la solución como se había diseñado.
  - Validar la opción de diseñar controles complementarios que bajen o disminuyan el riesgo.

- La vulnerabilidad está cerrada pero aparece un nueva. Un ejemplo de esto puede ser la identificación de un sistema operativo sin soporte, se realiza la migración a un sistema operativo soportado pero no se le aplican las actualizaciones de seguridad, por lo que de una vulnerabilidad podemos pasar a decenas o cientos. Cuando aparecen nuevas vulnerabilidades es necesario:
    - Validar que no son falsos positivos.
    - Elaborar un plan de remediación de vulnerabilidades y volver a verificar su cierre tras su implementación.

3.2. Hardening

Las aplicaciones utilizan software base que permiten su funcionamiento, por ejemplo:

- Sistema Operativo
- Servidor WEB
- Motor de base de datos
- Software de virtualización

El aseguramiento, endurecimiento, securización o hardening, es definido de al menos 2 formas:

- El proceso de asegurar el software base, disminuyendo así la superficie de ataque.
- La configuración más segura del software base.

Para realizar el hardening se utilizan guías de aseguramiento, las cuales han sido creadas:

- Por entidades u organizaciones dedicadas a este fin, como lo son CIS (Center for Internet Security) o la agencia nacional de seguridad (NSA) de los Estados Unidos.
- Por los propios fabricantes del software.

Una de las recomendaciones más importantes a la hora de realizar hardening es establecer y ejecutar un completo set de pruebas, pues en muchas ocasiones realizar el aseguramiento afecta las aplicaciones.

Observemos un extracto del contenido para una guía de aseguramiento desarrollada por CIS (Center For Internet Security, 2017), específicamente para el servidor IIS versión 10:

Lo anterior muestra una serie de controles a aplicar como parte del aseguramiento. Estos documentos son extensos y alcanzan a incluir cientos de controles.

Es posible encontrar en algunos modelos de seguridad en el ciclo de desarrollo que el hardening se realice en etapa de implementación, lo cual es también válido. Lo importante al final es que la actividad se ejecute previo paso de la aplicación a producción, de lo contrario estaríamos sacando a productivo un ambiente inseguro.

4. Seguridad en la etapa de mantenimiento

Entremos a las actividades de seguridad de la última etapa del ciclo de desarrollo en cascada. Estas actividades tienen en común que son operativas, por lo que se ejecutan más de una vez.

4.1. Gestión de Cambios

Una vez se ha liderado el software y se han surtido las etapas, es normal:

- Que se ejecuten modificaciones sobre la aplicación:
  - Mejoras
  - Solución de errores o bugs
  - Nuevos requerimientos funcionales o no funcionales

- Qué se pretenda mejorar la seguridad de la misma:
  - Luego de realizar un Pentesting
  - Luego de realizar un análisis de vulnerabilidades
  - Mejores prácticas

Todo esto se ve reflejado en cambios. Los cambios se deben controlar de acuerdo a un proceso definido y estructurado. El estándar ITIL (Biblioteca de Infraestructura de Tecnologías de Información), incluye un proceso de gestión de cambios, para el cual se manejan los siguientes conceptos (AXELOS, 2011):

- Cambio: modificación de un servicio de IT, en este caso una aplicación.
- RFC (Requerimiento de cambio): propuesta para la ejecución de un cambio.
- CAB (Comité de cambios): grupo interdisciplinario que autoriza la ejecución de un cambio.

El proceso de gestión de cambios genérico sigue la secuencia RFC -> CAB -> Implementación. Los roles principales que ejecutamos los profesionales de seguridad están asociados al CAB e incluyen:

Analizar los riesgos a nivel de seguridad que implique el cambio.

- Verificar el cumplimiento regulatorio a nivel de seguridad.
- Validar los temas de licenciamiento.
- Verificar que existe un procedimiento de rollback.
- Aprobar o denegar el cambio.

Por lo anterior, es importante que al menos un representante de seguridad haga parte del CAB.

¿Sabía qué...?

Los incidentes sobre las aplicaciones se deben manejar de acuerdo al procedimiento de gestión de incidentes que tenga definida la compañía.

En síntesis...

Las tres últimas etapas de seguridad en el ciclo de desarrollo y sus actividades son de igual o mayor importancia que las tres primeras.

Referencias

Center For Internet Security. (2017). *CIS Microsoft IIS 10 Benchmark*. www.cisecurity.org. Recuperado de https://downloads.cisecurity.org/

AXELOS. (2011). *Glosario y abreviaturas de ITIL Español (Latinoamericano)*. www.exin.jp. Recuperado de https://www.exin.jp/assets/exin/frameworks/108/glossaries/latam_spanish_glossary_v1.0_201404.pdf

INFORMACIÓN TÉCNICA

FACULTAD DE
INGENIERÍA, DISEÑO
E INNOVACIÓN

Módulo: Seguridad en el Ciclo de Desarrollo

Unidad 2: Etapas de seguridad en el ciclo de desarrollo de *software*

Escenario 4: Seguridad en las etapas de pruebas, despliegue y mantenimiento

Autor: Miguel Ángel Zambrano Puentes

Asesor Pedagógico: Edwin Mojica Quintero

Diseñador Gráfico: Brandon Steven Ramírez Carrero

Asistente: Ginna Quiroga