Implatación de técnicas y herramientas de pentesting en el proceso de desarrollo de software

Emilio J Roldán

25 de mayo de 2021

Índice general

Capítulo 1

Introducción.

1.1. motivación y objetivos

El motivo principal que me ha llevado a realizar este proyecto es relatar como realizar un proceso de pentesting resaltando dos herramientas que a día de hoy hay muchos pentester que no suelen utilizar como son los análisis de código, sobre todo la parte estática, así como la integración de dichas pruebas en el ciclo de desarrollo Software.

Capítulo 2

Análisis del estado del arte

2.1. Proceso de pentesting

2.1.1. ¿que es un prueba de penetración o pentest?

Según la definición de OWASP[1] un test de penetración o pentesting, a veces denominado prueba de caja negra, es esencialmente el arte de probar un sistema o aplicación para descubrir vulnerabilidades de seguridad, sin conocer el funcionamiento interno de la mismas. Normalmente el equipo encargado de las pruebas de penetración accede a las aplicaciones como si fuesen usuarios. El pentester tratará con que ese nivel de acceso encontrar vulnerabilidades que se puedan explotar en la aplicación.

El propósito de la prueba de penetración es determinar la presencia de vulnerabilidades potencialmente explotables y analizar el impacto de estas, sí se detecta alguna. La mejor forma de probar una defensa es tratando de penetrar en ella.

[1] **web1**.

2.1.2. Fases del prueba de intrusión

A la hora de realizar una prueba de intrusión o pentest distinguimos las siguientes fases, basándonos en la distinción realizada en pentesting con Kali;[2] dicjhas fases son las siguientes:

- Alcance y términos de la prueba de intrusión.
- Recolección de información.
- Análisis de vulnerabilidades.
- Explotación de vulnerabilidades.
- Postexplotación del sistema.
- Generación de informes.

Alcance y términos de la prueba de intrusión.

Para esta fase normalmente se genera un documento de plan de pruebas. En muchos casos es necesaria la revisión y aprobación de dicho documento por parte del dueño del sistema a probar (SUT), antes de poder comenzar con el proceso de pentesting. En dicho documento de pruebas se suel detallar la siguiente información:

- Sistema sobre el que se realizan las pruebas.
- Los tipos de prueba a realizar.
- Herramientas que se van a utilizar.
- Proceso de seguimiento de los defectos encontrados.
- Documentos que se entregarán durante el proceso de pentesting.
- Restricciones en la ejecución de la prueba de intrusión

Recolección de información.

Una vez definido el plan de pruebas procederemos a recolectar información del sistema o aplicación indicado en dicho plan. Principalmente obtendremos información mediante los procesos de enumeración y análisis de código que detallaremos en el siguiente capítulo.

[2] **ref1**.

Análisis de vulnerabilidades.

Al finalizar los procesos anteriores se analizarán los defectos encontrados para descartar falsos positivos y después se hará entrega un reporte de análisis dinámico con los defectos no descartados. Para cada uno de los defectos detectados que se incluyan en el reporte abriremos defecto en el sistema de gestión de defectos.

Explotación de vulnerabilidades.

En el caso de que uno o varios defectos necesiten ser explotados, y siempre solicitando permiso se detallará el proceso de explotación indicando las herramientas y exploits necesarios para realizar este proceso. En este proceso también se deben detallar las consecuencias, si las hubiese de la ejecución de las herramientas y exploits a utilizar sobre la aplicación o sistema objetivo.

Postexplotación del sistema.

En este caso también es necesario solicitar permiso al dueño del sistema, detallando la forma en que persistirá el ataque en la aplicación o sistema objetivo.

Generación de informes.

Llegados a este punto ya se deben haber hecho entrega de los reportes del análisis estático, si se dispone de acceso al código fuente, y del reporte de análisis dinámico. Sí se ejecutasen los procesos de explotación o Postexplotación se ampliaría el reporte de análisis dinámico con la información recabada en dichos procesos.

A parte de los reportes anteriores, se debe entregar un informe de resultado de pruebas con el resultado de ejecución del proceso de pentest incluyendo en el mismo el detalle de los defectos reportados en el sistema de gestión de defectos, si es posible, así como el estado en que se encuentran en el momento de entrega de dicho reporte.

2.2. Herramientas de enumeración

El proceso de enumeración trataremos de recabar información de recurso accesibles del sistema o aplicación. Para este proceso existen númerosas utilidades, entre las más utilizadas estarían:

- Nmap: Utilizada para el escaneo de puertos
- Nikto: Utilizada para recopilación de recursos disponibles.
- Dirb: Utilizada para recopilación de recursos disponibles.
- TestSSL: Utilizda para la revisión de certificados y mecanismos de cifrado SSL.

2.3. Herramientas de análisis de código

Detro de los procesos actuales de SSDLC cada vez cobran más importacia la inclusion de herramientas de analisis de codigo durante el proceso de desarrollo del Software.

Dentro de las herramientas de análisis de código, podemos hacer la siguietne distinción:

- **Herramientas de Análisis de código estático (SAST).** El análisis estático es un proceso que se realiza sobre el código de una aplicación sin necesidad de ejecutarse.

  El análisis de código estático, también conocido como Análisis de código fuente SCA, realiza pruebas sobre el código fuente para la detección temprana de defectos en dicho código. El uso de este tipo de herramientas es recomendable realizarlos en la fase de implementación del ciclo de desarrollo seguro SSDLC.

- **Herramientas de análisis de código Dinámico (DAST.)** Este tipo de análisis se realiza sobre una aplicación o servicio desplegado y en ejecución, a diferencia del tipo anterior.

  En análisis DAST enviará peticiones maliciosas al sistema objetivo para verificar la presencia de diversos tipos de ataques.

- **Herramientas híbridas.** Las herramientas hibridas son aquellas que presentan proceso para definir los dos tipos de análisis anteriores.

2.3.1. Herramientas de análisis estático de código

La metodología OWASP ASVS 4.0 se introdujo una sección para añadir los controles de codigo fuente como un requisito más dentro de la lista de requerimientos para un desarrollo seguro:

V1.10 Malicious Software Architectural Requirements

#	Description	L1	L2	L3	CWE
1.10.1	Verify that a source code control system is in use, with procedures to ensure that check-ins are accompanied by issues or change tickets. The source code control system should have access control and identifiable users to allow traceability of any changes.		✓	✓	284

Figura 2.1: ASVS 4.0 10.0.1 Security control.

Actualmete en mucho de los ciclos de desarrollo estas herramientas se encuentran integradas dentro de los procesos de Integración continua (CI) y despliegue continuo (CD), esto permite que ante cualquier cambio en el código se ejecuten estas herramientas de forma automática permitiendo que ante cualquier cambio se ejecuten este tipo de herramientas de forma automática en los procesos de compilación y despliegue.

También es común que las herramientas de análisis de código estén integradas dentro de los IDEs de desarrollo; lo cual permite que los desarrolladores también puedan hacer uso de estas herramientas y mejorar la calidad del código antes de su entrega.

Entre las distintas herramientas de análisis, para la implementación de nuestra infraestructura de pruebas haremos uso de las siguientes herramientas:

- SonarQube
- Dependency-check

SonarQube

Es una plataforma de para el análisis estático de código, dispone de distintos escáneres para la mayor parte de lenguajes de programación. Entre las versiones disponibles de SonarQube, podemos hacer uso de la versión *"Community"* que es de uso libre.

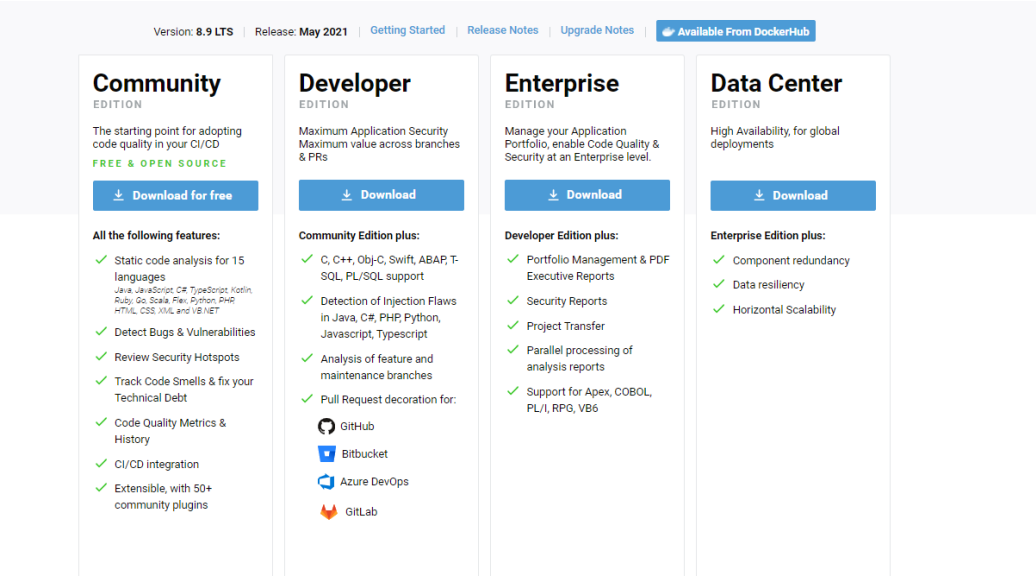

Figura 2.2: Versiones Sonarqube 8.9.

La versión *"Community"* incluye escáneres para los siguientes lenguajes de programación:

- Java
- JavaScript
- c#
- TypeScript
- Ruby
- Go
- Scala
- Flex
- Python
- PHP
- HTML
- CSS
- XML
- VB.Net

Además, mediante extensiones de la comunidad podemos añadir escáneres para los siguientes lenguajes:

- PL
  SQL
- C
  C++

Dependency-check

Es una herramienta de análisis de dependencias que intenta detectar vulnerabilidades divulgadas públicamente contenidas en las dependencias de un proyecto. Para ello, determina si existe un identificador de enumeración de plataforma común (CPE) para una dependencia determinada. Si lo encuentra, generará un informe vinculado a las entradas CVE asociadas.

2.3.2. Herramientas de análisis dinámico de código

Capítulo 3

Diseño solución técnica.

Capítulo 4

Ejecución casos de prueba.

4.1. Aplicación en desarrollo de aplicaciones Web

4.2. Aplicación en desarrollo de Servicios Web

Glosario

ASVS El proyecto OWASP Appplication Security Verification Standard (ASVS) proporciona una base para realizar los controles de seguridad técnicos en aplicaciones web, admás también proporciona un listado de requisitos a cumplir para un desarrollo seguro.. 7

CD Del inglés "Continous Deployement", término que hace referencia a la implantación de procesoa automáticos de desplige de las aplicaciones en el ciclo de vida del desarrolo de software.. 8

CI Del inglés *"Continous Integración"*, término que hace referencia a la implantación de procesos automáticos de compilación y revison del codigo fuente en el ciclo de vida del desarrollo software.. 8

CVE Del inglés *"Common Vulnerabilities and Exposure" (CVE)*, es una lista de información registrada sobre vulnerabilidades de seguridad conocidas, en la que cada referencia tiene un número de identificación CVE-ID.. 10

DAST Del inglés "Dynamic Application Security Testing", término que hace referencia a las pruebas de análisis dinámicas de código.. 7

exploit Término inglés que hace referencia a una secuencia de comandos utilizados para, aprovechándose de un fallo o vulnerabilidad en un sistema, provocar un comportamiento no deseado o imprevisto.. 5

OWASP El Open Web Application Security Project (OWASP) es una comunidad mundial libre y abierta enfocado en mejorar la seguridad del desarroollo de software. 7

SAST Del inglés "Static Application Security Testing", término que hace referencia a las pruebas de análisis estático de código. 7

SCA Del inglés "Static Code Analysys", término que hace referencia a las pruebas de análisis estático de código. 7

SSDLC Del inglés "Secure Software Development Life Cycle". El Software Development Life Cycle (SDLC) es un proceso de desarrollo estructurado enfocado en la producción de sofware de calidad, con el menor costo y el period más corto posible de tiempo. Un proceso seguro de SDLC, además añade procesos adicionales, encaminados a mejorar la

calidad del sofware, tales como pruebas de penetración, revisiones de código o análisis de dependencias.. 7

SUT Del inglés *"Sytem Under Test"*, término que hace referencia a la aplicación o sistema sobre el cual se ejecutarán las pruebas.. 4