0VERL0AD 1N TH3 N3T => Http://0verl0ad.blogspot.com

El fingerprinting dentro de la seguridad web

Por The X-C3LL

INDICE:

1. Introducción

La enumeración es una etapa básica en la realización de una auditoría de seguridad. Esta consiste en la recopilación de información sobre el objetivo que estamos auditando, con el fin de poder dirigir a posteriori un análisis más concreto. La información básica que debemos recopilar, dentro del marco de la seguridad a nivel web, tiene que poder contestar a la mayoría de estas preguntas:

- **¿Qué aplicaciones web corren?**
- **¿Y en qué versiones?**
- **¿Hay rutas o archivos que pudieran considerarse como sensibles?**
- **¿Sobre qué servidor trabaja?**

Como se puede observar en el índice, al principio del artículo, no voy a hablar de nada que no sea nuevo, por lo que salvo algunas cosas que quizás sí sean algo poco conocido o curioso quizás resulte algo aburrido. La idea de crear este artículo es recopilar la máxima información sobre este tema, tanto para poder tenerla toda en un mismo artículo, como para que pueda ser de utilidad a todos aquellos que empiezan en este mundillo.

2. Identificación de aplicaciones web

La identificación de aplicaciones webs se puede realizar por distintos métodos, tanto manuales como automáticos, siendo en la mayoría de los casos imprescindible la aplicación de varios de ellos para que el resultado final sea real.

Cada vez se está disminuyendo más la exposición explícita del nombre de la aplicación (y a veces de la versión) en los footers, dandose más casos donde los propios WebMasters eliminan estos textos tipo “Powered by ...” o directamente los spoofean, poniendo versión más antiguas o con vulnerabilidades bastante conocidas.

Pese a que esta técnica de ocultación básica por oscuridad a primera vista pueda parecer inútil y poco efectiva, en realidad si se mira desde una perspectiva más pragmática se puede ver como una medida básica que te bloquea a toda una horda de Script Kiddies que únicamente buscan webs vulnerables a través de Google, Bing u otros motores de búsquedas, por lo que si no te indexan la cadena de “Powered by...” se reducen la posibilidades de que lancen un ataque contra el sitio web.

2.1. Información dentro de HTML

Como ya dijimos que los footers con información es algo que se está perdiendo paulatinamente debemos de adoptar otros métodos para la determinación de la aplicación que está corriendo, por lo que siempre deberemos de analizar el documento HTML que visualiza nuestro navegador.

Las etiquetas <Meta> pueden ser una fuente de información bastante interesante, no son pocos los casos donde aparece el nombre de la aplicación (ejemplos claros son Joomla, Wordpress). Por otra parte los enlaces, imágenes, formularios y scripts presentes pueden ser también marca identificatoria, puesto que algunas rutas y archivos son propios de determinadas aplicaciones webs (**/wp-content/** , **/component/**, etc.); además pueden aparecer en ellos rutas o archivos que puedan considerarse como sensibles (como zonas de administración, por ejemplo).

Esta técnica puede automatizarse utilizando crawlers que naveguen por todo el sitio web, almacenen el documento HTML y después le pasen un patrón de búsquedas para buscar coincidencias con cadenas previamente conocidas.

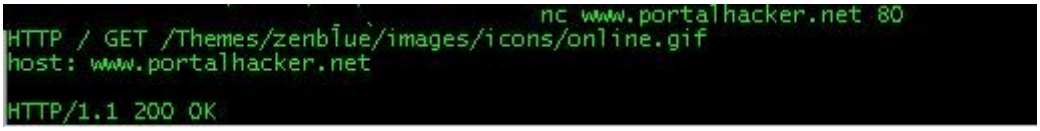
2.2. Códigos de estado HTTP

Los códigos de estado HTTP están formados por 3 dígitos y están agrupados en 5 categorías (indicada cada una por el primer dígito del código). Estos códigos son devueltos al inicio de las cabeceras HTTP de respuesta del servidor, y como su propio nombre indican, informan del status. Los códigos probablemente más conocidos son los 200 (OK), 404 (File Not Found), 403 (Forbbiden).

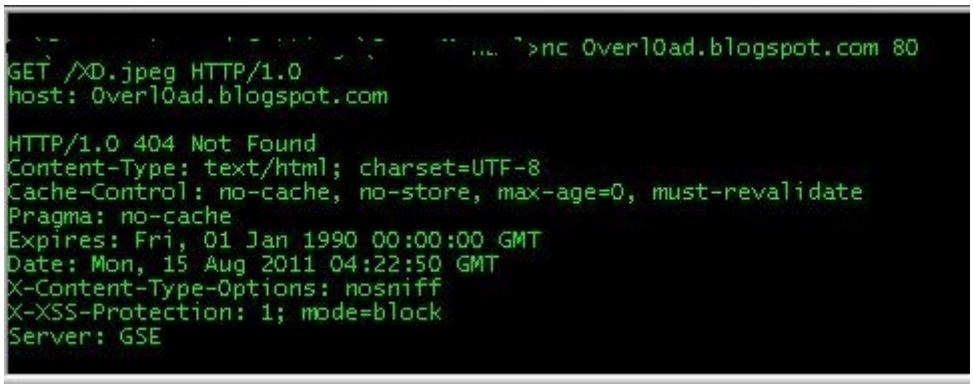
Según las indicaciones estandarizadas en el RFC 2616, sección 6.1 y desde 10.1 en adelante, los códigos 200 deberán de ser enviados en el caso de que la petición haya tenido éxito, mientras que el 404 se envía cuando no encuentra nada que concuerde con la petición URI, siendo lo normal que aparezca cuando un archivo al que le hemos hecho una petición no exista.

Para poder trabajar con cabeceras HTTP existe un amplio abanico de herramientas que nos facilitan el trabajo, aunque en esta ocasión vamos a utilizar la herramienta base para cualquier conexión: NetCat.

Si hacemos una petición con NetCat a un archivo existente, lo más común es que nos aparezca algo tipo:



Mientras que si lanzamos la petición GET hacia un archivo inexistente, el servidor nos responderá con un 404:



Con esta base podemos pensar en automatizar un script que lance peticiones GET a archivos y rutas que tenemos almacenadas en un diccionario; el contenido de dicho diccionario consistiría en una relación de rutas y archivos propios de cada aplicación web. Al analizar las respuestas del servidor ante las peticiones GET podremos tomar los 200 como “positivo” para ese archivo y los 404 como “negativo”, de esta manera conseguimos concretar el tipo de aplicación, y en algunos casos hasta la versión.

A continuación dejo el código de un ejemplo bastante tosco que acabo de codear rápidamente y que no sé hasta que punto pueda funcionar correctamente. Partiendo de este concepto de automatizar las peticiones y discriminar las respuestas positivas -además de tener un buen diccionario- prodremos averiguar qué aplicación web está corriendo, e incluso podríamos adaptarlo para averiguar qué plugins. Como bien digo, el código es un simple PoC para que situeis por donde pueden ir los tiros más o menos.

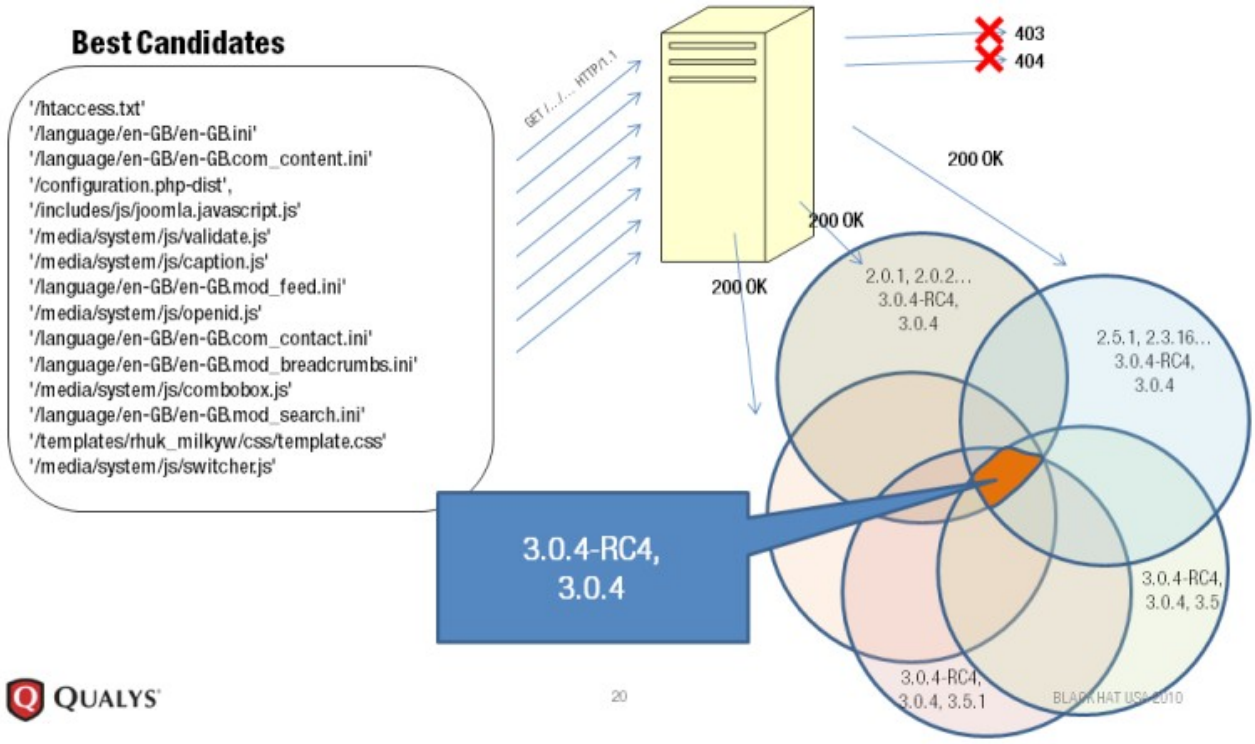
```
# PoC en perl para enumeración de aplicaciones
# Detecta estados 200

use IO::Socket;

my $host = $ARGV[0];
my $dic = $ARGV[1];
my $total = 0;
my $positivo = 0;

open (DICCIONARIO, $dic) || "Error: no se puede abrir el fichero";
foreach $uri (<DICCIONARIO>) {

  $total++;
  $socket = IO::Socket::INET->new(
    Proto => "tcp",
    PeerAddr => "$host",
    PeerPort => "80",
    ) || die "Error: No se puede conectar";

  print $socket "GET $uri HTTP/1.1\nHost: $host\n\n";
  @Respuesta = <$socket>;
  if ($Respuesta[0] =~ /200/ ) {
     $positivo++;
 }
  close($socket)
}
close(DICCIONARIO);

print q(
+----------------------+
| PoC para fingerprint  |
+----------------------+

);
print "Se han realizado el test para $dic con $total peticiones,
$positivo positivos";
```

Ésta técnica es utilizada para listar los plugins de WordPress, discriminando entre códigos 404 (no existe) y cualquier otro (400, 403), usando este script para nmap (http://seclists.org/nmap-dev/2011/q1/att-806/http-wp-plugins.nse). Contiene una serie de opciones para agilizar el brute force, como por ejemplo un listado de los 100 plugins más comunes.

2.3. Checksums

Los checksums permiten comprobar si un archivo ha sido modificado o no. Al convertir en un hash (md5, SHA1, etc.) el contenido del archivo conseguimos tener un string identificatorio de nuestro archivo, ya que si tan sólo se modificase un sólo byte de nuestro archivo el hash cambiaría, por lo que si tenemos el hash original y el generado posterior podemos comprobar si ha habido alguna alteración.

Este alto grado de identificación de archivos puede utilizarse también para realizar fingerprinting. La idea es calcular los checksums de los archivos típicos de una aplicación o plugin que no sufran modificación en condiciones normales (como imágenes, .js, css, etc.) y almacenar los hashes creados dentro de una base de datos local. Después se lanza una aplicación que descargue esos archivos estáticos y calcule su hash también. El resto sería simplemente comprobar coincidencias entre los hashes hasta encontrar qué está corriendo en el server remoto.



(Imagen extraida de la web de BlindElephant)

Actualmente esta técnica ha sido implementada con éxito por la herramienta Blind Elephant, codeada en python. Esta herramienta es bastante interesante por el planteamiento del que parte para trabajar, comprobando como ya dijimos los checksums de archivos estáticos para ir acotando de qué aplicación y de qué versión se trata.

2.4. Motores de búsquedas

Una opción menos invasiva, pero que también puede ser útil en ocasiones, es la de realizar búsquedas de archivos, y al hacer esto encontrar también rutas, usando motores como Google o Bing. Es muy común que estos motores de búsqueda indexen archivos claves para la identificación de una aplicación web, por lo que se pueden utilizar para buscarlos sin llegar a entrar dentro de la web. Se trata por tanto de un método bastante menos invasivo que todos los anteriores porque no deja rastro (cualquier automatismo de los explicados anteriormente deja mucho ruido en los logs).

Es muy importante hacer hincapié en la necesidad de combinar varios motores de búsqueda y no basarse sólo en uno. Como la interpretación del archivo robots.txt no está estandarizada, cada motor toma su propia directiva de como debe actuar ante las restricciones. Esto genera una situación en la cual X motor no ha indexado unos cuantos archivos que puedan aportar luz en el fingerprinting, pero el motor Y sí. Al combinar los resultados de ambas búsquedas podremos identificar de forma más exacta la aplicación.

2.5. Robots.txt

A la inversa de lo comentado en el anterior epígrafe, donde buscábamos archivos que habían escapado a las restricciones de indexación de robots.txt, lo que podemos hacer (si existe este archivo, claro está) es echar un ojo en él. En muchas ocasiones también podremos encontrar ahí rutas que sean interesantes para poder identificar la aplicación.

```
User-agent: *
Disallow: /adcentric
Disallow: /adinterax
Disallow: /atlas
Disallow: /doubleclick
Disallow: /eyereturn
Disallow: /eyewonder
Disallow: /klipmart
Disallow: /pointroll
Disallow: /smartadserver
Disallow: /unicast
Disallow: /viewpoint
Disallow: /addineyeV2.html
Disallow: /canvas.html
Disallow: /DARTIframe.html
Disallow: /interim.html
Disallow: /oggiPlayerLoader.htm
Disallow: /videoeggbackup.html

Disallow: /facebook_xd_receiver.html
Disallow: /readme.html
Disallow: /rpc_relay.html
Disallow: /twitterlists/
Disallow: /wp-content/plugins/
Disallow: /wp-content/themes/
```

Como se puede observar en el ejemplo de arriba podemos ver que en robots.txt han incluido información que puede ayudarnos a determinar que tiene instalado un WordPress (**/wp-content/**).

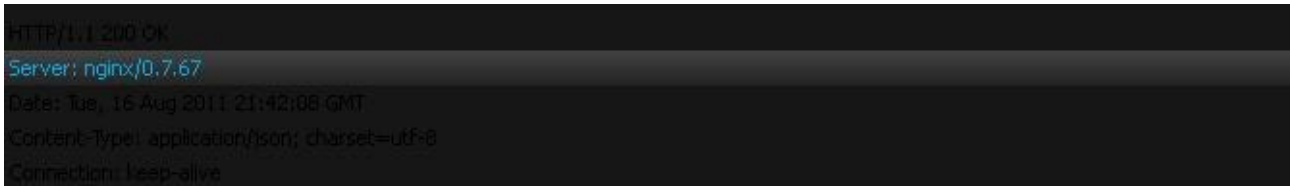
3. Identificación de Servidores Web

Tanto en una auditoría de seguridad como en un ataque dirigido es necesario conocer el servidor web sobre el que está montado. Para realizar tal reconocimiento existe también un abanico de técnicas, con mayor o menor grado de fiabilidad, a nuestro disposición que pasaremos a comentar a continuación.

3.1. Análisis de cabeceras HTTP

La técnica más sencilla de identificación es la lectura del banner desde las cabeceras HTTP de respuesta. Dentro de dicha cabecera suele aparecer un campo de nombre **Server** cuyo valor informa de las características básicas como el tipo de servidor (Apache, IIS...), los intérpretes que tiene (PHP, ASP, Perl...) y en algunos casos los mods instalados.

Para poder visualizar esta información podemos recurrir a cualquier herramienta que permita la lectura de las cabeceras HTTP, aunque por comodidad recomendamos el add-on para Firefox Live HTTP Headers (el hecho de tenerlo directamente en el navegador facilita mucho las cosas). Un ejemplo de banner típico es este:



En este caso podemos ver como tiene instalado un servidor **nginx** en su versión **0.7.67.**

Pero no es oro todo lo que reluce. El problema de los banners es que pueden ser spoofeados fácilmente a través de la propia configuración del servidor, confundiendo así al auditor, o símplemente actuando como HoneyPot para los Script Kiddies -en vez del banner original del servidor se puede poner uno de otro tipo de servidor con versiones obsoletas para las cuales haya exploits conocidos, como por ejemplo IIS 5.0, bastante famoso debido a las revistas HackXCrack-.

Las formas de spoofear son variadas, desde modificar **SecServerSignature** en mod_security hasta comodificar los archivos antes de compilarlos. Es por esta facilidad de ocultación por la que no podemos fiarnos únicamente del banner, sino que debemos de emplear otros métodos para realizar un fingerprinting lo más certero posible.

En general, en cualquier campo en el que se realice fingerprinting la base es la misma: analizar las diferencias entre las respuestas que ofrecen los diferentes entes que estamos analizando. En el caso de las cabeceras HTTP los puntos interesantes a analizar son los mensajes de error, el orden en que aparecen dichas cabeceras, y el comportamiento ante distintos métodos HTTP.

HTTPrecon es una herramienta fruto de un proyecto cuya esencia radica en lo comentado anteriormente: realizar fingerprinting del servidor analizando las cabeceras HTTP. Tiene una base de datos bastante extensa donde tienen almacenados los distintos comportamientos propios de cada servidor y versión. De esta forma esta herramienta se convierte en indispensable para la auditoría.

3.2. HTTP Parameter Pollution / Contamination

Hace relativamente poco se ha abierto un nuevo campo de investigación dentro de la seguridad web a raiz de los artículos “HTTP Parameter Pollution” y más reciente aún “HTTP Parameter Contamination” -nos referiremos a estas técnicas como **HP(P/C)**- . Lo que se exponía en ambos artículos era cuan diferente reaccionaban los servidores ante peticiones corruptas, como por ejemplo setear dos veces una misma variable en una petición (**?va1=A&var1=B**) o utilizar caracteres reservados (**?var1=&X-C3LL**).

Como ningún servidor está preparado para esta situación anómala cada cual reacciona de una forma distinta, y es por ello que uno de los posibles usos de HP(P/C) sea la de fingerprinting. Desconozco si en este momento se ha desarrollado, o se está desarrollando, alguna herramienta que automatice la tarea, así que deberemos de hacerlo manualmente. A continuación dejo una pequeña relación (imágenes extraidas de las publicaciones anteriormente mencionadas) de comportamientos ya conocidos:

Technology/HTTP back-end	Overall Parsing Result	Example
ASP.NET/IIS	All occurrences of the specific parameter	par1=val1,val2
ASP/IIS	All occurrences of the specific parameter	par1=val1,val2
PHP/Apache	Last occurrence	par1=val2
PHP/Zeus	Last occurrence	par1=val2
JSP,Servlet/Apache Tomcat	First occurrence	par1=val1
JSP,Servlet/Oracle Application Server 10g	First occurrence	par1=val1
JSP,Servlet/Jetty	First occurrence	par1=val1
IBM Lotus Domino	Last occurrence	par1=val2
IBM HTTP Server	First occurrence	par1=val1
mod_perl,libapreq2/Apache	First occurrence	par1=val1
Perl CGI/Apache	First occurrence	par1=val1
mod_perl,lib???/Apache	Becomes an array	ARRAY(0x8b9059c)
mod_wsgi (Python)/Apache	First occurrence	par1=val1
Python/Zope	Becomes an array	['val1', 'val2']
IceWarp	Last occurrence	par1=val2
AXIS 2400	All occurrences of the specific parameter	par1=val1,val2
Linksys Wireless-G PTZ Internet Camera	Last occurrence	par1=val2
Ricoh Aficio 1022 Printer	First occurrence	par1=val1
webcamXP PRO	First occurrence	par1=val1
DBMan	All occurrences of the specific parameter	par1=val1~~val2

Query string	Web Servers response / GET values			Explanation
	Apache/2.2.16 / PHP/5.3.3	Tomcat 6 / JSP	IIS 6 / ASP	
?test[1=2	test_1=2	test[1=2	test[1=2	Curly bracket is changed with underscore
?test.1=2	test_1=2	test.1=2	test.1=2	Curly bracket is changed with underscore
?[1&d=2	d=2	[1 / d=2	[1&d=2	First is ignored whole param, second query delimeter
?1[]xx=2	1=array(2)	1[]xx=2	1[]xx=2	Characters beetwen array and equal sign are ignored
?test+d=1+2	test_d=1 2	test d=1 2	test d=1 2	First sign plus converted to underscore, second to space
?test d=1 2	test_d=1 2	test d=1 2	test d=1 2	First space converted to underscore
?test=%	test=%	NULL	test=	JSP ignore param, ASP ignore value
?test%x=1	test%x=1	NULL	testx=1	JSP ignore param, ASP ignore procent sign
?test%00=1	test=1	test=1	test=1	JSP include NULL value in param key
?test%00a=1	test=1	test=1	test=1	JSP include NULL value in param key, others ignore after
?test=1%001	NULL	test=1=1	test=1	Apache ignore param, JSP inlcude NULL, ASP ignore after
?%00=1	NULL	NULL=1	NULL	JSP have strange behaviour, doesn't print key
	Debian		W2K3	

4. Referencias

[1] Web Aplication Fingerprinting
http://www.exploit-db.com/download_pdf/17538
[2] HTTP al descubierto
http://www.portalhacker.net/index.php/topic,120908.0.html
[3] Blind Elephant
http://blindelephant.sourceforge.net/
[4] HTTPrecon Project
http://www.computec.ch/projekte/httprecon/
[5] HTTP Parameter Contamination
http://netsec.rs/files/Http%20Parameter%20Contamination%20-%20Ivan%20Markovic%20NSS.pdf
[6]HTTP Parameter Pollution
https://www.owasp.org/images/b/ba/AppsecEU09_CarettoniDiPaola_v0.8.pdf