SE2B1.- Enumerar medidas de protección en entornos cloud.

Tecnologías en Aplicaciones web.

Warm-Up

Marina Navarro Pina

Contenido

Licencia

1. Protocolo HTTP

Es el protocolo de comunicaciones utilizado para acceder a la World Wide Web y es usado por todas las aplicaciones web. Fue diseñado originalmente para obtener recursos estáticos basados en texto. Ha sido extendido para permitir el soporte de aplicaciones distribuidas.

HTTP es un protocolo cliente-servidor, lo que significa que el cliente envía un mensaje de solicitud al servidor y espera un mensaje de respuesta del servidor. Es un protocolo sin estado, lo que significa que el servidor no guarda información del cliente, cada petición es independiente de las demás.

Fue diseñado para transmitir HTML (HyperText Markup Language) pero hoy en día se utiliza para transmitir todo tipo de documentos (imágenes, audio, video, PDF, etc.) y para crear Aplicaciones Web.

El documento completo del protocolo HTTP lo puedes encontrar en https://tools.ietf.org/html/rfc2616.

1.1- Mensajes HTTP

Un mensaje HTTP (no importa si es de petición o respuesta) se compone de 3 partes:

- La primera línea (que es diferente para la petición y la respuesta).
- Los encabezados.
- El cuerpo (opcional)

Veamos un ejemplo de un mensaje de petición (sin cuerpo):

```
GET /index.html HTTP/1.1
Host: wikipedia.org
Accept: text/html
```

En este ejemplo estamos solicitando el recurso `/index.html` de wikipedia.org. La primera línea del mensaje de petición se compone de:

- El verbo (en este caso `GET`)
- El recurso (en este caso `/index`)
- La versión de HTTP (en este caso `HTTP/1.1`)

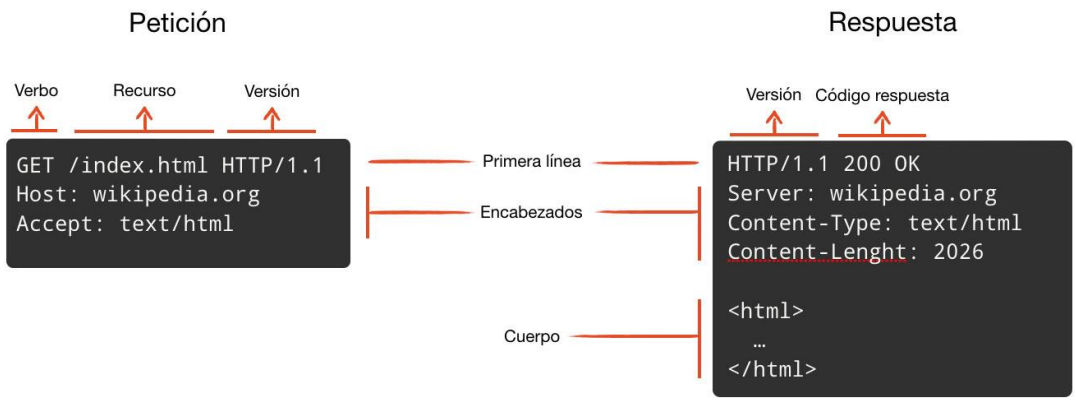
Ahora veamos un ejemplo de un mensaje de respuesta:

```
HTTP/1.1 200 OK
Server: wikipedia.org
Content-Type: text/html
Content-Lenght: 2026
<html>
  …
</html>
```

La primera línea del mensaje de respuesta se compone de:

- La versión de HTTP (`HTTP/1.1`)
- El código de respuesta (`200 OK`)

La siguiente imagen muestra mejor las partes de cada mensaje:



1.1-1. El verbo HTTP

La primera línea de un mensaje de petición empieza con un **verbo** (también se le conoce como **método**). Los verbos definen la *acción que se quiere realizar sobre el recurso*. Los verbos más comunes son:

- **GET**: se utiliza para *solicitar* un recurso.
- **POST**: se utiliza para *publicar* un recurso.
- **PUT**: se utiliza para *reemplazar* un recurso.
- **DELETE**: se utiliza para *eliminar* un recurso.

Existen otros (https://developer.mozilla.org/es/docs/Web/HTTP/Methods) pero estos son los más comunes.

Cuando ingresas a una página desde un navegador, por debajo el navegador envía un mensaje GET, lo mismo cuando oprimes un vínculo a otra página.

1.1-2. El código de respuesta

La primera línea de un mensaje de respuesta tiene un código de 3 dígitos que le indica al cliente cómo interpretar la respuesta.

Los códigos de respuesta se dividen en cinco categorías dependiendo del dígito con el que inician:

- 1XX: Información
- 2XX: Éxito
- 3XX: Redirección
- 4XX: Error en el cliente
- 5XX: Error en el servidor

Seguramente estás familiarizado/a con el famoso error 404 que retornan los servidores cuando el recurso no fue encontrado. O con el error 500 cuando ocurre un error en el servidor. Pero existen muchos más: https://developer.mozilla.org/es/docs/Web/HTTP/Status

1.1-3. Las cabeceras - *encabezados*

Las cabeceras brindan información adicional sobre la petición o la respuesta. Tienen la siguiente sintaxis:

```
[nombre del encabezado]: [valor del encabezado]
```

Cabeceras comunes incluyen:

- `Content-Type`: el tipo de contenido que se está enviando en el cuerpo de un mensaje, por ejemplo `text/html`.
- `Accept`: el tipo de contenido que el cliente está esperando.
- `User-Agent`: el tipo de navegador que está haciendo la petición.
- `Connection`: Le indica a la otra parte el final de la comunicación.
- `Content-Encoding`: Tipo de codificación utilizada por el contenido del cuerpo del mensaje como "gzip".
- `Content-Length`: Longitud del cuerpo del mensaje en bytes.
- `Transfer-Encoding`: Especifica cualquier codificación que fue realizada sobre el cuerpo del mensaje para facilitar la transferencia.

Además, se tienen *Cabeceras de Solicitud* y *Cabeceras de Respuesta*: https://developer.mozilla.org/es/docs/Web/HTTP/Headers

Para enviar cabeceras falsas se debe haber modificado esta información antes de visitar la página.

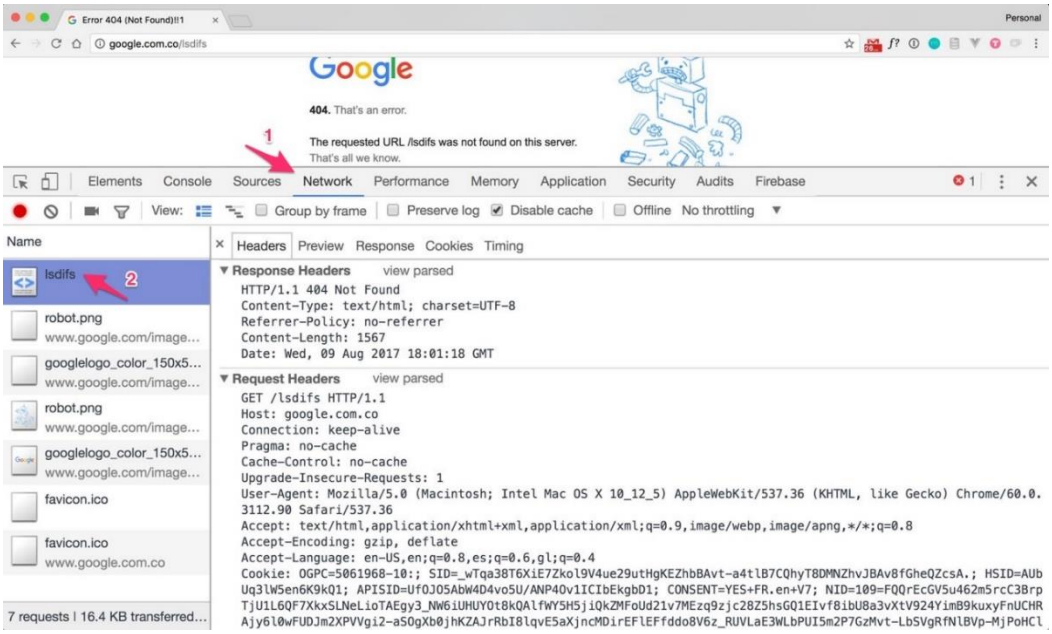
Lista de User Agents:

https://perishablepress.com/list-all-user-agents-top-search-engines

User-Agent-Switcher:

https://addons.mozilla.org/es/firefox/addon/uaswitcher/

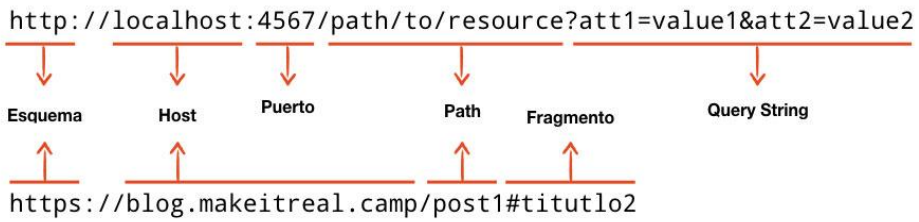
La siguiente imagen muestra, en las herramientas de desarrollo de Chrome, el mensaje de petición y respuesta cuando hacemos una petición a http://google.com.co/lsdifs. Fíjate que el código de respuesta es `404 Not Found`:



1.2- URL

Un **URL** (Uniform Resource Locator) se utiliza para ubicar un recurso en Internet. Los URLs no solo se pueden utilizar para el protocolo HTTP, se utilizan en muchos otros protocolos.

La siguiente imagen muestra las partes de un URL utilizando dos ejemplos:



- **Esquema**: El esquema define el protocolo a utilizar, para HTTP puede ser `http` o `https` (el protocolo seguro de HTTP).
- **Host**: La IP o el nombre del servidor que se quiere acceder (127.0.0.1, localhost, google.com, www.google.es, etc.)
- **Puerto**: El puerto en el que está escuchando el servidor HTTP. Si se omite se asume que es el 80.
- **Path**: La ruta al recurso que se quiere acceder.
- **Query String**: Contiene información adicional para el servidor en forma de propiedades (`atributo=valor`). Las propiedades se separan por `&`.
- **Fragmento**: La referencia a una ubicación interna del documento.

1.3- Cookies

Son una parte clave del protocolo HTTP. El mecanismo de las cookies permite al servidor enviar ítems de datos al cliente, el cual el cliente almacena y reenvía al servidor. Estas son reenviadas en cada petición posterior sin ninguna acción requerida por la aplicación o usuario.

El servidor define una cookie con "`Set-Cookie`" en la cabecera de respuesta. Y el navegador web automáticamente añade "`Cookie`" en todas las cabeceras de solicitud.

https://developer.mozilla.org/es/docs/Web/HTTP/Cookies

1.4- HTTPS

Es esencialmente el mismo protocolo HTTP pero puesto en un túnel como mecanismo de seguridad en el transporte (SSL). Esto protege la integridad y privacidad de los datos que transitan sobre la red, reduciendo las posibilidades de ataques de interceptación. Las solicitudes y respuestas HTTP funcionan exactamente de la misma manera, sin importar si se utiliza SSL para el transporte.

1.5- Autenticación HTTP

El protocolo HTTP incluye sus propios mecanismos para la autenticación de usuarios utilizando varios esquemas de autenticación.

- `Basic`: Mecanismo más simple, que envía las credenciales del usuario como una cadena codificada en Base64, en la cabecera de solicitud.
- `NTLM`: Mecanismo de reto/respuesta y utiliza una versión del protocolo Windows NTLM.
- `Digest`: Mecanismo de reto/respuesta y utiliza sumas de verificación MD5 con las credenciales del usuario.

https://developer.mozilla.org/es/docs/Web/HTTP/Authentication

Es inusual encontrar estos protocolos de autenticación en Internet, son más comunes en aplicaciones web de intranets.

2. Funcionalidad Web

Las Aplicaciones Web emplean varias tecnologías para proporcionar su funcionalidad. Puede emplear docenas de distintas tecnologías dentro de los componentes del servidor y el cliente.

2.1- Funcionalidad del lado del Servidor

Cuando un cliente solicita un recurso dinámico, la respuesta del servidor es creada al vuelo, y cada usuario recibe contenido único personalizado. Estos son generados por scripts o códigos ejecutados en el lado del servidor. Entre las tecnologías utilizadas en el lado del servidor se tienen:

- Lenguajes de scripting: como PHP, VBScript y Perl
- Plataformas de aplicaciones web: como ASP.NET y Java
- Servidores Web: como Apache, IIS.
- Bases de Datos: como MongoDB, Oracle, MySQL.
- Otros componentes back-end: como sistemas de archivos, web services y servicios de directorio.

2.2- Funcionalidad del lado del Cliente

Para que la aplicación en el lado del servidor pueda presentar resultados, requiere recibir entradas y acciones del usuario. Como se realizan mediante un navegador estas comparten tecnologías comunes:

- HTML: Es un lenguaje basado en marcas (tags), para describir una estructura.
- Hiperenlaces: La mayoría de la comunicación del cliente al servidor se conduce con hiperenlaces.
- Formularios: Mecanismo que permite a los usuarios ingresar entradas arbitrarias mediante un navegador.
- CSS (Cascading Style Sheets): Lenguaje para describir la presentación de un documento escrito en un lenguaje de marcas.
- JavaScript: Lenguaje de programación utilizado para extender las interfaces web de formas no posibles utilizando solo HTML.
- Ajax: Colección de técnicas de programación utilizadas en el lado del cliente para crear interfaces de usuario que imiten la forma de aplicaciones de escritorio.
- HTML5: Actualización del HTML estándar.

2.2-1. Detección de eventos JavaScript

Una técnica utilizada en las pruebas, es tratar de evadir el código JavaScript que la aplicación espera sea ejecutada en el navegador. Para esto, se requiere buscar algunos de los siguientes eventos:

- `onClick`: Cuando se hace clic en un enlace o elemento de formulario.
- `onMouseOver`: Cuando se mueve el punto fuera de un link o área.
- `onFocus`: Cuando el foco de la entrada es un elemento de formulario.
- `onBlur`: Cuando el foco se retira de un elemento del formulario.
- `onLoad`: Cuando una página se carga en el navegador.
- `onSubmit`: Cuando se envía un formulario.

Etiquetas a observar:

`<body>, <form>, <select>, <checkbox>, <img>`

2.3- Sesiones

Las aplicaciones necesitan rastrear el estado de la interacción del usuario con la aplicación a través de varias peticiones. Por ejemplo, una aplicación de compras puede permitir al usuario navegar por un catálogo de productos, añadir artículos, ver y actualizar los contenidos, proceder con la compra y dar información personal y detalles del pago.

En algunas aplicaciones la información de estado se almacena en un componente del cliente en lugar del servidor, los datos se pasan al cliente en cada respuesta desde el servidor y son enviados de regreso al servidor en cada solicitud del cliente. Y como es obvio, el cliente puede modificar esta información de estado.

Las peticiones HTTP son stateless por naturaleza. Esto significa que cada petición realizada es independiente, incluso si se realiza desde la misma computadora por una misma persona.

A fin de guardar datos con las preferencias de los visitantes se utilizan las **cookies**. Estas cookies se guardan en el *navegador web de cada visitante* (usualmente como petición del servidor, a través de un header llamado `Set-Cookie`).

Las cookies generalmente están limitadas por un tamaño máximo. Así mismo, están expuestas en el lado del cliente (las cookies son client-side).

Una solución ante esto último involucra tanto el uso de sesiones como el uso de cookies. Las **sesiones** se guardan en el servidor, pero se asocian a una cookie creada en el cliente (navegador web). Esta cookie permite al navegador web mostrar su identificación ante el servidor, y que este reconozca sus datos.

La imagen muestra como, en un primer momento, el servidor envía una cabecera en la que solicita que el cliente almacene una cookie que contiene el identificador de sesión y, posteriormente, el navegador incluye esta cookie en la respuesta para que el servidor reconozca que la petición pertenece a una determinada sesión.

Otra ventaja del uso de los identificadores de sesión es que permiten que la información asociada a la sesión esté almacenada en el servidor, un entorno de seguridad más controlado, al que no tiene acceso el cliente directamente.

3. Codificación

Las aplicaciones web emplean diferentes tipos de esquemas de codificación para los datos. HTTP y HTML se basan en texto, mediante la utilización de diferentes esquemas de codificación, se asegura que se puedan manejar caracteres inusuales o datos binarios.

3.1- Codificación URL

Utilizado para codificar caracteres problemáticos del conjunto de caracteres ASCII extendido, para poder ser transportados con HTTP. La codificación inicia con "%" seguido del código ASCII de dos dígitos del carácter en hexadecimal.

`%3d` -> = `%20` -> "Un espacio" `%00` -> "Byte Nulo"

3.2- Codificación Unicode

Unicode está diseñado para soportar todos los sistemas de escritura del mundo. La codificación Unicode de 16 bits funciona igual que la codificación URL.

`%u2215` -> /

UTF-8 es un estándar de codificación de longitud variable que emplea uno o más bytes para expresar un carácter.

`%c2%a9` -> @

3.3- Codificación HTML

Utilizado para representar caracteres problemáticos en un documento HTML. Varios caracteres tienen un significado especial como metacaracteres en HTML y se utilizan para definir la estructura del documento en lugar de su contenido.

`&quot;` -> " `&amp` -> & `<` -> <

Cualquier carácter puede ser codificado en HTML utilizado su código ASCII en decimal.

`&#34` -> "

3.4- Codificación Base64

Permite representar cualquier dato binario utilizando solo caracteres ASCII imprimibles. Es utilizado para codificar las credenciales de usuario en la autenticación BASIC. Si el bloque final de datos de entrada resulta menor que tres trozos de datos de salida, la salida se rellena con uno o más "=".

`QWxvbnNvIENhYmFsbGVybw==`

Algunas aplicaciones web usan codificación Base64 para transmitir datos binarios dentro de las cookies o parámetros y ofuscarlos.

3.5- Hexadecimal (Base 16)

Sus dígitos incluyen los números del 0 al 9 y los caracteres abarcan desde la letra "A" hasta la letra "F". Todos en mayúsculas o minúsculas y rara vez se verán mezcladas.

`0x00` `0xFF` `0x41` (Espacio) `0x20` (A)

3.6- Octal (Base 8)

Bastante inusual a diferencia de otras bases (16, 36, 64). No utiliza ninguna letra y sus dígitos abarcan desde el número "0" al número "7".

`017` (15 en Decimal)

3.7- Base 36

Es un híbrido entre la Base 16 y Base 64, inicia en "0" y recorre todo el alfabeto luego del número "9". No se detiene en la letra F, por lo que incluye todas las letras desde la "A" hasta la "Z". No distingue entre mayúsculas o minúsculas y no incluye signos de puntuación.

`BZH6R` (20131011 en Decimal)

3.8- Hash

Es una función matemática de una sola dirección. Dada una entrada de cualquier tamaño, se produce exactamente una salida de un tamaño fijo.

Tiene dos importantes propiedades:

- Dado un hash, es difícil encontrar el dato de entrada que lo produce.
- Dada una entrada, es difícil encontrar otra entrada que produzca el mismo hash.

MD5: (32 Caracteres Hexadecimales) df02589a2e826924a5c0b94ae4335329

(24 caracteres en Base 64) PlnPFeQx5Jj+uwRfh//RSw==

SHA-1: (40 Caracteres) bc93f9c45642995b5566e64742de38563b365a1e

(28 caracteres en Base 64) 9EkBWUsXoiwtICqaZp2+VbZaZdI=