Departamento de Ciencias de la Computación
Facultad de Ciencias Físicas y Matemáticas
Universidad de Chile
CC4303-2 Redes

INFORME ACTIVIDAD HTTP

Construir un Proxy

Alumno: Benjamín San Martín
Profesora: Ivana Bachmann
Auxiliar: Vicente Videla

Fecha de entrega: 8 de Septiembre de 2023
Santiago, Chile

Para esta actividad, se solicitó la construcción de un Proxy, que actúa como intermediario entre el cliente y el servidor final. El proceso se dividió en dos partes: la construcción del servidor y la construcción del proxy. La idea era que este último funcionara como un tipo de control parental capaz de bloquear ciertas páginas web y censurar palabras prohibidas, además de agregar headers que fueran capaces de identificar al cliente que solicitó la página web.

1. Flujo de funcionamiento

Para comprender el flujo de funcionamiento, se solicitó la construcción de un diagrama que ejemplificara dicho proceso. El diagrama es el siguiente:

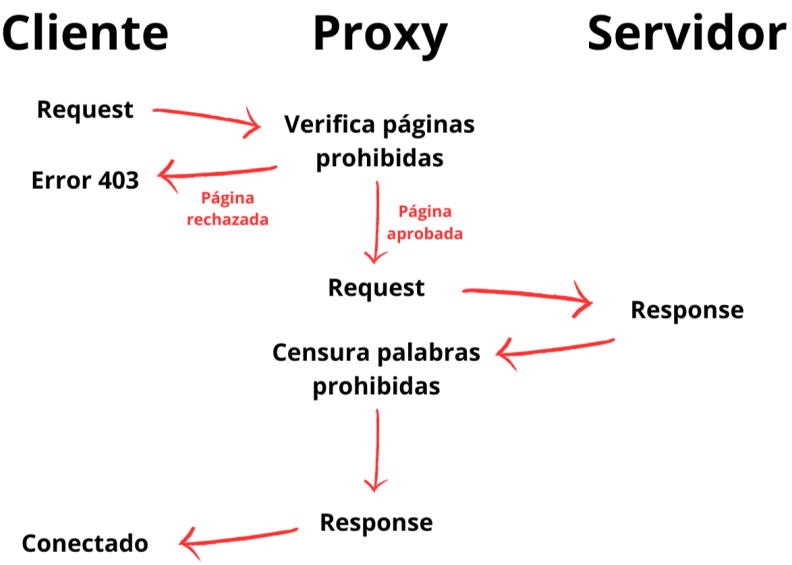

Figura 1: Diagrama de flujo del funcionamiento

En el diagrama se puede observar cómo el cliente efectúa una solicitud (request) que es enviada al proxy. El proxy verifica si el cliente está intentando acceder a una página con restricciones; si esto es cierto, devuelve un mensaje de error 403. En caso contrario, dirige la solicitud al servidor final (página web). El servidor final devuelve una respuesta (response) en la cual el proxy busca y censura palabras prohibidas (si las hubiera). Posteriormente, el proxy envía esta respuesta modificada de vuelta al cliente.

El flujo de implementación es el siguiente: En primer lugar, se crean variables globales para definir el tamaño del búfer, la secuencia de término (‘b’\r\n\r\n’‘) y la dirección del proxy, que será "localhost.[en] el puerto 8000. Luego, se crea un socket de conexión y este comienza a recibir conexiones de los clientes a través de la función ‘accept()‘. Una vez se recibe el mensaje del cliente, se utiliza la función ‘parse_HTTP_message()‘ para dividirlo en encabezados y cuerpo. Se verifica

que la línea de inicio (startline) no contenga URLs bloqueadas. A partir de aquí, se presentan dos opciones posibles:

1. Si no se trata de una URL bloqueada, se extrae la URL mediante la función 'get_url()'. Luego, se crea un nuevo socket de conexión que enviará la solicitud del cliente al servidor final. Este nuevo socket se conecta a la URL utilizando el puerto 80. Con la función 'create_HTTP_message()', se genera un mensaje que incluye tanto los encabezados como el cuerpo del mensaje. Una vez que el mensaje es construido, se envía al servidor.

   A continuación, se procesa la respuesta recibida del servidor. Se verifica que no haya palabras prohibidas en la respuesta, de ser así, son reemplazadas por censuras. Finalmente, se envía la respuesta procesada de vuelta al cliente y se cierra la conexión usando 'close()'.

2. Si la página está bloqueada, se utiliza la función 'create_HTTP_message()' con un parámetro distinto para generar un mensaje de error que incluya la URL solicitada por el cliente. Luego, se envía este mensaje de error al cliente y se cierra la conexión.

2. Ejecución del código

El código debe ser ejecutado a través de una terminal y requiere un archivo JSON que contenga los siguientes valores:

```
"user_name": "Su_Usuario",
"user": "Su_Email",
"blocked": ["http://www.dcc.uchile.cl/", "http://cc4303.bachmann.cl/secret"],
"forbidden_words": [{"proxy": "[REDACTED]"}, {"DCC": "[FORBIDDEN]"},
                    {"biblioteca": "[???]"}]
```

Para ejecutar el código, abre una terminal en el directorio que contiene los archivos 'proxy.py' y el archivo JSON, y luego ingresa el siguiente comando:

```
python proxy.py –config archivo.json
```

Una vez realizado esto, el proxy estará en funcionamiento. Para ponerlo a prueba, puedes utilizar los siguientes comandos:

```
curl example.com -x localhost:8000
curl cc4303.bachmann.cl -x localhost:8000
curl cc4303.bachmann.cl/replace -x localhost:8000
curl cc4303.bachmann.cl/secret -x localhost:8000
```

A continuación, se procederá a comentar los resultados obtenidos en cada una de estas páginas web.

1. En el caso de example.com, el sitio web no presenta ningún tipo de restricción, por lo que visualizarlo a través del proxy es lo mismo a verlo de manera normal.

2. En el caso de cc4303.bachmann.cl, al visualizarlo a través del proxy, el mensaje en la página es distinto de de verlo sin el proxy. Debido a que el proxy actúa como intermediario y agrega

el header de ”X-ElQuePregunta” por lo tanto la página muestra el nombre de usuario. Sin embargo, cuando se accede sin el proxy, se muestra el mensaje ”Bienvenide ... oh? no puedo ver tu nombre :c!”. Tambien la palabra ”proxy” está censurada por lo que el proxy la cambia por [REDACTED].

3. En cc4303.bachmann.cl/replace, hay bastantes cambios debido a la censura de las palabras prohibidas. La palabra "proxy.[es] reemplazada por "[REDACTED]", "DCC.[es] reemplazada por "[FORBIDEN]", y finalmente "biblioteca.[es] reemplazada por "[???]".

4. En cc4303.bachmann.cl/secret es una página prohibida por lo que el acceso está bloqueado por el proxy y al intentar ingresar se muestra el error 403.

3. Decisiones de diseño

A continuación, se comentarán las razones detrás de ciertas decisiones tomadas al momento de diseñar el código:

- **Uso de diccionarios:** La función ‘parse_HTTP_message()‘ retorna una lista con dos elementos, siendo el primero de ellos un diccionario. Se optó por un diccionario para almacenar los headers, ya que estos no siguen un orden específico. Buscar un header en una lista iba a ser lento y deficiente. Por lo que, al utilizar un diccionario, se asigna el nombre del encabezado como clave y su valor como el valor correspondiente. Esto permite un acceso directo al valor del header simplemente buscándolo por su nombre.

- **Lista para almacenar el mensaje :** La elección de trabajar con una lista de dos elementos (un par ordenado cumpliría la misma función) tenía el propósito de separar las dos partes del mensaje de manera fácil. Esto permite trabajar fácilmente con cualquiera de las partes del mensaje.

- **Estructura de ‘contains_end_of_message()‘:** Esta función retorna una lista que incluye la posición de la secuencia de fin y la diferencia entre el final del mensaje y la posición de la secuencia. Esta diferencia es bastante importante ya que, si su valor es 0, la secuencia de terminación se encuentra al final del mensaje. En cambio, si el valor no es 0, significa que la secuencia está en medio del mensaje. Esta distinción es importante porque si la secuencia no está al final, existen bytes del body que se encuentran en conjunto con los encabezados. En este caso, no hay que contar estos bytes para obtener el resto del mensaje.

- **Función ‘create_HTTP_message()‘:** Esta función acepta la estructura que entrega ‘parse_HTTP_message()‘ y entrega un único mensaje. Además, agrega el encabezado "X-ElQuePregunta". Ademas, si el valor de la variable ïs_forbiden.[es] verdadero entonces la página está prohibida y la función retorna un mensaje de error correspondiente.

A continuación se van a responder las preguntas del enunciado:

- **¿Cómo sé si llegó el mensaje completo? :** Si el tamaño del body es igual al que está en el header Content-length, si el mensaje llega cortado pero el tamaño es igual entonces el problema ya no es la recepción del mensaje sino el header.

- **¿Qué pasa si los headers no caben en mi buffer? :** Si no caben entonces guardará lo que tiene de mensaje en otra variable y seguirá recibiendo mensajes hasta detectar que el mensaje total guardado en la otra variable contiene la secuencia de termino.
- **¿Cómo sé que el HEAD llegó completo?¿Y el BODY?:** El head va a estar completo cuando el mensaje contenga la secuencia de termino (‘b’\r\n\r\n’‘), si no la contiene el programa se quedará estancado en un ciclo. El Body sabemos que llego completo por el header Content-length.