

Trabajo Práctico

Proxy POP3

Protocolos de Comunicación

Segundo Cuatrimestre 2018

Grupo 3

Integrantes:

Della Sala Rocío - 56507

Giorgi, María Florencia - 56239

Rodríguez, Ariel Andrés - 56030

Santoflaminio, Alejandro - 57042

1. Índice

Protocolo

Aplicaciones desarrolladas

Problemas encontrados

Diseño

Implementación

Limitaciones de la aplicación

Posibles extensiones

Conclusiones

Ejemplos de prueba

Guía de instalación

Instrucciones para la configuración

Ejemplos de configuración y monitoreo

Documento de diseño del proyecto

2. Descripción

2.1 Protocolo

Definimos el protocolo de administración entre el proxy y el cliente de configuración a partir de sus características más generales, para luego definir las funcionalidades y su estructura.

**Características**

En primer lugar, establecimos que queríamos implementar un protocolo orientado a caracteres, pero como la conexión entre el administrador y proxy es SCTP, nos dimos cuenta que era conveniente aprovechar el uso de datagramas para enviar las tareas a ejecutar. Finalmente, optamos por un *protocolo orientado a bit*. Este tipo de protocolo nos permitirá hacer un parseo más sencillo de los comandos en el proxy.

En segundo lugar, definimos el mecanismo de comunicación entre ambos siendo este *request/response*. De este modo se definen dos tipos de mensajes:

* Los tipo request que envían tareas a realizar como por ejemplo setear el programa externo o pedir información de una métrica.
* Los tipo response que envían la respuesta de ejecutar la tarea que el request específico. Dependen del tipo de request que se envió.

Por último, el protocolo es orientado a conexión debido a que mantenemos una sesión del cliente. Esto se realizó mediante la autenticación del mismo. Esto es similar a POP3 ya que se tendrán dos estados: uno de autenticación y otro de transacción.

**Funcionalidades (comandos)**

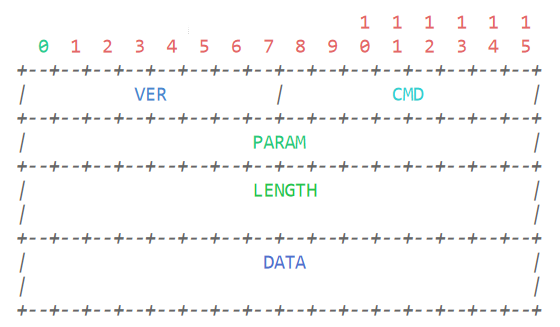
Se pensaron los comandos que el usuario pueda ejecutar según los requerimientos pero también en base a otras cuestiones como la seguridad del proxy, su futuro testeo, su independencia con programas externos entre otras cuestiones. Cada uno de los comandos tiene una posible respuesta, que incluye un estado y un mensaje de respuesta si es necesario.

* Autorización
  + AUTH user pass
  + Argumentos: Se envían como argumentos el nombre de usuario y la contraseña.
  + Necesitamos autenticación, debido a que es conectado a sesión. Esto nos permitirá evitar controles como por ejemplo que el administrador nunca intentará cambiar el programa transformador por un comando bash que pueda ser peligroso para el funcionamiento del proxy (rm).
* Obtener una métrica
  + GET\_METRIC metric
  + Argumentos: Se envía como argumento una constante que define la métrica a utilizar.
  + Especificar aca que devuelve bytes transferidos.
* Apagar/Prender la transformación
  + SWITCH\_TRANS on/off
  + Argumentos: Se envía como argumento una constante que define si esta prendido o apagado.
* Obtener el programa externo de transformación
  + GET\_TRANS
  + Argumentos: No tiene argumentos.
  + Útil para que la implementación del proxy no quede restringida al correcto funcionamiento del programa de transformación. Además nos permite probar con otros programas ademas del mimeFilter para detectar si los errores (si los hay) provienen del funcionamiento del proxy o del programa externo.
* Setear el programa externo de transformación
  + SET\_TRANS trans
  + Argumentos: Se envía como argumento el programa transformador.
  + Comando
* Obtener los mimes prohibidos
  + GET\_MIME
  + Argumentos: No tiene argumentos.
  + Comando para obtener los mimes que están siendo censurados por el programa transformador.
* Agregar un mime a la lista de mimes prohibidos
  + FORBID mime
  + Argumentos: Se envía como argumento el mime a agregar de la lista de mimes censurados
  + Comando para agregar un mime a la lista de los mimes que estan siendo censurados por el programa transformador. Permite prohibir todo un tipo, indicando “\*” como subtipo. Por ejemplo si hacemos FORBID image/\* prohibiremos todos los tipos image.
* Quitar un mime de la lista de mimes prohibidos
  + ALLOW mime
  + Argumentos: Se envía como argumento el mime a sacar de la lista de mimes censurados.
  + Comando para remover un mime de la lista de los mimes que estan siendo censurados por el programa transformador.
  + Aclaración: Si teníamos censurado todo un tipo, no podemos luego permitir un subtipo de ese tipo. Es decir, si teníamos censurado image/\*, no podemos hacer ALLOW image/png, debemos hacer primero ALLOW image/\* y luego prohibir los subtipos deseados.
* Cerrar la sesión
  + QUIT
  + Argumentos: No tiene argumentos.

**Estructura del mensaje**

A partir de las funcionalidades que deseabamos implementar, definimos la estructura del request

Request



VER: Un byte que detalla la versión del protocolo. Se agrega para poder pensar el protocolo escalable a otras versiones y mantener retrocompatibilidad con clientes con versiones más viejas.

CMD: Un byte que detalla el comando a ejecutar. Estos estarán definidos como constantes donde:

0 Autenticarse

1 Obtener una métrica

2 Apagar/Prender la transformación

3 Obtener el programa externo de transformación

4 Setear el programa externo de transformación

5 Obtener los mimes prohibidos

6 Agregar un mime a la lista de mimes prohibidos

7 Quitar un mime de la lista de mimes prohibidos

8 Cerrar la sesión

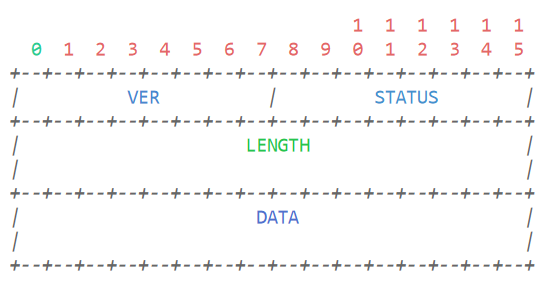
PARAM: 2 bytes para especificar los parámetros de los cuales se trata

LENGTH: 4 bytes que indican (de existir un mensaje) el tamaño del mismo.

DATA: Mensaje de tamaño variable indicado parámetros de tipo texto.

Response

Para la respuesta algunos campos se mantienen igual que la estructura del request.



STATUS: 1 byte para especificar el estado de la respuesta que puede ser +OK indicando que la tarea se realizó satisfactoriamente o -ERR para indicar un mensaje de error

2.2 Aplicaciones desarrolladas

En el proyecto encontramos tres aplicaciones desarrolladas: el servidor proxy para el protocolo POP3, un cliente para configurar el servidor proxy y un programa cuya tarea es filtrar media types de un mail.

Completar

3. Problemas encontrados

Se presentaron varias dificultades durante la realización del trabajo:

* Completar
* Falta de conocimiento sobre los distintos tipos de formato que pueden presentarse en un mail.
* Problemas para filtrar un mail anidado dentro de otro. Finalmente se resolvió mediante el manejo de stack de boundaries.
* Un error donde los archivos de los casos de prueba estaban corrompidos e impedían el correcto funcionamiento del TP.

3.1 Diseño

COMpletar

3.2 Implementación

Completar

Stripmime:

Para la realización de este programa se hizo uso de los parches provistos por la cátedra en campus y se trabajó a partir de ellos.

Requiere de las variables de entorno FILTER\_MEDIAS y FILTER\_MESSAGE. De esta forma, en el programa se realiza un listado donde se almacenan los tipos con sus respectivos subtipos en forma de nodos a partir de la información recibida en FILTER\_MEDIAS.

Se hace uso de una estructura para mantener el estado durante el parseo del mail, la cual contiene diferentes parsers para interpretar las diferentes partes de un correo.

Además de parsers, en esta estructura también almacenamos un stack de delimitadores correspondientes a los valores que tenemos como boundaries en un correo. Esto permite el filtrado de contenido en mensajes anidados.

Si no recibe ningún mensaje a reemplazar en la variable de entorno correspondiente, se reemplazará el contenido a filtrar con el texto “Parte reemplazada.”.

4. Limitaciones de la aplicación

Existen varias limitaciones en el proyecto.

COMPLETAR

Dentro de stripmime tenemos:

* La aplicación no puede procesar los comentarios dentro de un mail.
* La aplicación no funciona bien si se censura alguno de los tipos multipart.

5. Posibles extensiones

Completar

Dentro de stripmime podríamos agregar crear parser para los comentarios y alterar el funcionamiento para que censure correctamente los tipos multipart.

6. Conclusiones

Completar

Con respecto a stripmime corriendo el archivo grande de prueba (i\_big.mbox) pudimos obtener los siguientes tiempos para los siguientes comandos:

1. Primero filtrando un media type no presente en el archivo:

export FILTER\_MEDIAS=a/b

time cat i\_big.mbox | ./stripmime | pv > out

real 1m33.663s

user 1m28.835s

sys 0m4.509s

1. Luego filtrando un media type presente en el archivo, cuyo contenido era extenso.

export FILTER\_MEDIAS=application/x-cd-image

time cat i\_big.mbox | ./stripmime | pv > out

real 1m21.334s

user 1m21.096s

sys 0m1.604s

Podemos compararlo con el tiempo que se obtiene al realizar cat con el mismo archivo:

time cat ../test\_cases/i\_big.mbox > out

real 0m9.727s

user 0m0.000s

sys 0m0.587s

7. Ejemplos de prueba

Comandos a ejecutar desde el directorio root del proyecto.

**Stripmime:**

El siguiente comando filtrará los tipos image del archivo ii\_images.mbox y la salida se podrá ver en el archivo out generado en el directorio root.

$ FILTER\_MEDIAS=image/\* ./stripmime <../test\_cases/ii\_images.mbox> out

8. Guía de instalación

8.1 Compilación de proxypop3

8.2 Compilación de administrador

8.3 Compilación de mimeFilter

9. Instrucciones de configuración