Informe TPE protocolos de comunicación

Rodrigo Manuel Navarro Lajous

Agustín calatayud

Sebastián Favarón

Francisco Delgado

Instituto Tecnológico de Buenos Aires

Resumen

Este documento informa sobre el trabajo realizado para el trabajo práctico especial de la materia Protocolos de Comunicación por parte del grupo 7 del segundo cuatrimestre de 2018. Se encontrará una descripción completa del trabajo realizado junto con el análisis del mismo

1. Aplicaciones y protocolos desarrollados

1.1. Protocolo SPCP

1.1.1. Introducción al protocolo

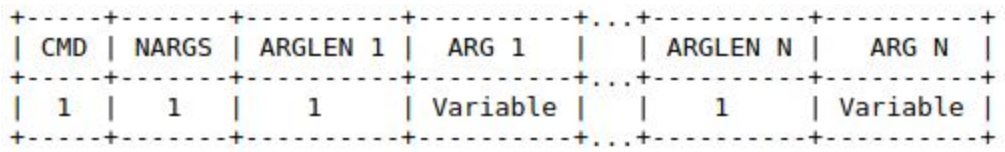
El Simple Proxy Communication Protocol(SPCP) es un protocolo binario utilizado para poder configurar y obtener métricas de un servidor proxy. En el diseño del protocolo se priorizo la simpleza así permitiendo que sea implementado lo más fácilmente posible siempre y cuando no comprometa el objetivo principal de administrar el proxy

1.1.2. Operación Básica

Inicialmente, el servidor spcp inicializa su servicio escuchando en el puerto 9090 por default para aceptar conexiones sctp entrantes. Cuando el cliente desea utilizar el servicio este debe establecer una conexión sctp con el servidor. Una vez establecida la conexión se procede a la etapa de autenticación, el cliente manda su usuario y su contraseña y estos son validados por el servidor. Una vez autenticado se procede al estado de transacción en donde se mandan requests tanto para modificar el estado del proxy como para obtener métricas del mismo recibiendo para cada uno una respuesta con la información en el caso de que se haya podido obtener, o un estado de error apropiado. Finalmente se envía un request de terminación en donde se le indica al servidor que el cliente desea finalizar su sesión y el servidor da por finalizada la sesión.

1.1.3. El request

La estructura de un request

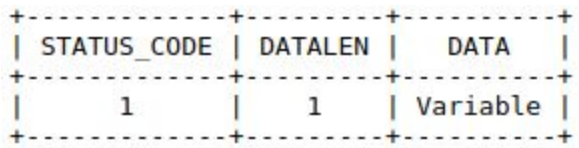


El request se compone por un número variable de segmentos, estos son:

- CMD: El identificador de comando, tiene un tamaño de un byte.
- NARGS: La cantidad de argumentos, tiene tamaño de un byte.
- ARGLEN: Cada argumento se encuentra precedido por el largo del mismo, este tiene un máximo de 255 ya que este campo tiene un byte de longitud.
- ARG: El argumento de tamaño anteriormente especificado, los argumentos son representaciones ASCII.

1.1.4. La Response

La estructura de un response



El response se compone por un número variable de segmentos, estos son:

- STATUS_CODE: Identifica el código de la respuesta.
- DATALEN: La longitud de los datos de la respuesta del servidor.
- DATA: Los datos enviados por el servidor, estos se encuentran en codificación ASCII.

Discutimos brevemente los posibles códigos de respuesta que un servidor puede enviar a un cliente

- 0x00: es el response de OK, indica que el request se ejecutó con éxito por parte del servidor.
- 0x01: Error de autenticación, indica que hubo un error al tratar de autenticar el usuario.
- 0x02: Comando inválido, el comando que se envio en el request no es un comando válido.
- 0x03: Argumentos inválidos, indica que los argumentos para el comando no son válidos.
- 0x04: Error general, es el error de fallback del servidor cuando no puede especificar por que fallo.

1.1.5. Estado de autenticación

Una vez inicializada la conexión el servidor se encuentra en el estado de autenticación. El cliente debe proveer su usuario mediante el comando 0x00, al cual llamaremos USER.

El comando USER se compone de la siguiente manera:
Argumentos: uno, la representación ASCII del nombre de usuario.
Restricciones: Solo se puede utilizar en el estado de autenticación una vez inicializada la conexión o inmediatamente después de un USER o PASS erróneo.
Discusión: Primero se debe mandar el comando USER, si el usuario está registrado entonces el servidor responde con una respuesta positiva mandando el byte 0x00. En caso contrario responderá con un 0x01 (AUTH ERROR)

Inmediatamente después del comando USER debe proveer mediante otro request la contraseña asociada al usuario. Para esto emite un request al servidor con el comando 0x01, al cual le llamaremos PASS de ahora en adelante.

El comando PASS se compone de la siguiente manera:

Argumentos: uno, la representación ASCII de la contraseña.

Restricciones: Solo se puede utilizar en el estado de autenticación una vez inicializada la conexión e inmediatamente después de un USER.

1.1.6. Estado de transacción

Ya habiendo sido autenticado el servidor procede al estado de transacción, en este estado se podrá consultar el estado del proxy como también alterarlo. Para esto se usan distintos comandos que procederemos a describir:

0x02, conexiones concurrentes:

Argumentos: ninguno

Restricciones: Solo puede ser utilizado en el estado de transacción.

Discusión: El comando obtiene la representación ASCII de la cantidad de conexiones al servidor que se encuentran establecidas

0x03, bytes transferidos:

Argumentos: ninguno.

Restricciones: Solo puede ser utilizado en el estado de transacción.

Discusión: El comando obtiene como respuesta positiva la representación ASCII de la cantidad de bytes que fueron transferidos desde que el servidor arrancó. En el caso de que el servidor no pueda responder satisfactoriamente al request este responderá con un estado de error apropiado.

0x04, Accesos históricos:

Argumentos: ninguno.

Restricciones: Solo puede ser utilizado en el estado de transacción.

Discusión: El comando obtiene como respuesta positiva el número de accesos que hubieron al servidor desde que se inicializó. En el caso de que el servidor no pueda responder satisfactoriamente al request este responderá con un estado de error apropiado.

0x05, Transformación activa:

Argumentos: ninguno.

Restricciones: Solo puede ser utilizado en el estado de transacción.

Discusión: El comando obtiene como respuesta la representación ASCII del programa de transformación que está ejecutando el servidor. En el caso de no tener ningún programa de transformación se recibe una respuesta positiva sin ningún dato. Si por alguna razón no se pudo devolver la transformación activa entonces se devuelve un código de error apropiado como se especificó anteriormente

0x06, Modificar el tamaño de los buffers:

Argumentos: uno, la representación ASCII en decimal del nuevo tamaño para los buffers.

Restricciones: Solo puede ser utilizado en el estado de transacción.

Discusión: El comando cambia el tamaño que se le asignan a los buffers al momento de inicializar una nueva conexión al servidor. Si se pudo realizar el cambio con éxito entonces se obtiene como respuesta 0x00 (OK). En caso contrario se obtiene una respuesta con el error apropiado como se especificó anteriormente.

0x07, Cambio de transformación:

Argumentos: uno, la representación ASCII del nuevo comando de transformación.

Restricciones: Solo puede ser utilizado en el estado de transacción.

Discusión: El comando cambia la transformación a realizar. En el caso de querer desactivar las transformaciones simplemente se envía el comando de cambio de transformación sin datos. El servidor responde de manera positiva (0x00) cuando pudo cambiar con éxito las transformaciones, caso contrario responde con un error apropiado.

0x09, Terminación de la sesión:

Argumentos: ninguno

Restricciones: Solo puede ser utilizado en el estado de transacción.

Discusión: Cuando se envía un request con este comando se le avisa al servidor que el cliente desea finalizar la sesión, el servidor responde de manera exitosa y termina la conexión.

1.1.7. Limitaciones del protocolo

El protocolo, dada su simpleza, posee varias limitaciones y las discutimos a continuación

1.1.7.1. Tamaño de los argumentos

El protocolo posee un byte para indicar el tamaño del argumento, por lo que permite solamente argumentos de como máximo 255 bytes, esto no limita de manera sustancial el cambio del tamaño de los buffers ya que resultaría excesivo un buffer con un tamaño superior a 10^255 bytes. en cambio limita considerablemente la posibilidad de las transformaciones a realizar ya que la transformación debe ser un comando con una longitud máxima de 255 caracteres.

1.2. Servidor SPCP

1.2.1. Introducción al servidor SPCP

El proxy cuenta con un servidor SPCP implementado dentro del mismo proxy. Este servidor Implementa el lado servidor del protocolo SPCP para poder brindar monitoreo y métricas a los usuarios administradores del proxy.

1.2.2. Implementación del servidor

El servidor SPCP se implementa de manera no bloqueante y concurrente para poder permitir la ágil utilización del mismo por parte de múltiples administradores. Para esto se utiliza una estructura de servidor similar a la que se presenta en el servidor SocksV5. Se utiliza la estrategia de un selector el cual atiende a los file descriptors de los sockets mediante la función pselect.

El servidor cuenta con un socket pasivo SCTP el cual escucha por defecto en el puerto 9090 pero puede ser alterado mediante los argumentos pasados al proxy en el momento de la inicialización. Este socket pasivo establece las conexiones entre el servidor SPCP y el cliente. Una vez establecida la conexión se procede al manejo de los requests y las responses.

Se le da especial consideración al manejo de la autenticación dentro del servidor SPCP, ya que si bien el protocolo establece el mecanismo de autenticación, es el servidor quien decide cómo implementarlo. Para esto el servidor mantiene los nombres y contraseñas de los usuarios que tienen autoridad de administrar y en base a estos datos autentica al usuario y procede a atender los requests del mismo.

1.3. Cliente SPCP

El trabajo práctico cuenta con un cliente para poder demostrar las funcionalidades del servidor SPCP, este es relativamente simple pero permite al usuario utilizar toda la potencia del protocolo. El cliente permite especificar la IP y el puerto del servidor SPCP al cual desea conectarse.

Al iniciar el cliente éste intenta conectarse con el servidor. De ser exitoso entonces procede a pedirle al usuario que se autentique mediante un usuario y contraseña válidos. una vez autenticado se le presentan todas las opciones que puede elegir para interactuar con el servidor spcp. Puede realizar cuantas operaciones desee y terminar su sesión mediante el comando quit.

1.4. Proxy POP3

Al centro del trabajo práctico se encuentra el proxy pop3. Se encarga de interactuar con el servidor origen y el cliente. El proxy le permite al cliente poder acceder a su casilla de correo POP3 del servidor origen y beneficiarse del proxy. El cliente cuenta con soporte de pipelining independientemente de la implementación del servidor origen y además todos sus mensajes pasan por el proceso de transformación del proxy si este se encuentra activado.

1.4.1. Resolución de nombres

El proxy cuenta con la capacidad de hacer resolucion de nombres sin bloquearse, así un usuario del proxy no necesita conocer la ip de su servidor de origen sino que con su domain name basta. La resolución de nombres por su naturaleza es bloqueante, ya que debe realizar una consulta de DNS, es por esto que al momento de resolver una dirección esto se realiza en un thread aparte para que no se bloquee el proxy.

1.4.2. Determinación de pipelining

Como el proxy implementa pipelining independientemente de si el servidor origen lo implementa o no este debe poder detectarlo. Esto lo hace mediante un request de CAPA que es parseado para ver si tiene pipelining en su respuesta. Si el servidor origen no soporta el comando CAPA se tiene en cuenta que no tiene pipelining.

1.4.3. Política de logging

Para los logs del servidor se optó por logear cada request y response de los usuarios sin logear la totalidad de los mismos, si se sabe el nombre del usuario logueado se identifica al mismo en el log, pero por defecto es user unknown. Así podemos mantener constancia sobre que requests manda que usuario y las responses que le llegan y a que usuario pertenecen.

1.4.4. Implementación de pipelining con servidor no pipeliner

La forma en la que se realiza esto es recibiendo del cliente los requests pero enviando solamente uno al servidor origen para luego al retornar con una respuesta enviarle el siguiente comando que envió el cliente.

1.4.5. Manejo del request response

Para el manejo del request response se parsean las requests formando una cola de las mismas y enviando de a una, si el origen no tiene pipelining, o de a varias. Una vez enviadas se pasa a response, en aquella etapa se parsean las respuestas del servidor, si la respuesta es un mensaje y las transformaciones están activas se pasa al estado transformación. Luego de transmitirse las respuestas al usuario se pasa al estado request nuevamente.

1.4.6. Manejo de las transformaciones

Para las transformaciones lo que se hace es enviar la primera linea tal cual al cliente, si es positiva y se debe transformar se abren los pipes y el programa transformador. Una vez hecho esto, todo pasa por el transformador, siendo el cierre del pipe el final de la transferencia en ambos casos. Una vez transferida toda la respuesta al cliente se vuelve al estado correspondiente, ya sea request o response.

1.5. Censurador de MIME types

El censurador de MIME types, llamado stripmime, es un programa desarrollado por nosotros que invoca internamente el proxy (si este está configurado para hacerlo) siempre que este recibe un mail del servidor. Una vez que se lo llama, el stripmime hace uso de las variables de entorno FILTER_MEDIAS (que es un string con los MIMEs a filtrar separados por comas) y FILTER_MSG (otro string que contiene el mensaje con el que se reemplazan los MIMEs a filtrar) y el mail que ésta en un buffer (que puede recibir por medio de un archivo o de STD_IN) y lleva a cabo el filtrado de dicho mail, reemplazando donde encuentra un MIME a filtrar por el FILTER_MSG, cambiando el valor del header “Content-Type” a text/plain y el valor de “Content-Transfer-Encoding” a quoted-printable (para asegurarse que el mensaje escrito sea legible). Como stripmime lee mails desescapados (es decir, el dot-stuffing fue previamente removido) el cierre del pipe marca la finalización de mail y asi puede leer mensajes donde el body contiene una marca de finalización clasica de mensaje (\r\n.\r\n) dentro suyo.

2. Problemas encontrados

2.1. Determinación de pipelining

Uno de los requisitos del trabajo práctico era poder soportar pipelining incluso cuando el servidor de origen no era pipeliner.

Al principio consideramos la opción de hacerlo con dos requests pipelined de prueba pero resultaba problemático porque no sabríamos como se comporta un servidor pop3 si se le mandan comandos por pipeline sin que este lo soporte ya que no está explicitado en el RFC1939 [POP3]

Por esta razón optamos luego por emitir un request de capa al servidor origen y así detectar pipelining.

2.2. Detección y transformación de mensajes

Inicialmente el proxy actuaba como un proxy transparente en donde solamente escribía en el origen lo que recibe del cliente y escribía en el cliente lo que recibía en el origen. Esto funcionaba bien pero al momento de transformar, o de saber si la respuesta es multilínea o unilinea para poder manejar apropiadamente la respuesta resultaba imposible. Es por esto que se tuvo que hacer un parser de request en donde este mantiene una cola de requests. Esta cola de requests posee información relevante al proxy tal como la longitud del mismo, que request es, y si tiene argumentos o no. Es particularmente importante saber qué request es y si tiene argumentos o no ya que esto permite que sepamos si la respuesta se tiene que transformar.

2.3. Pipelining con servidor de origen no pipeliner

Si el servidor de origen no es pipeliner no podíamos mandar todos los request que recibimos del cliente al servidor origen. Como explicamos previamente obtenemos el largo del request y la información necesaria para detectar el final de su response (si es multi o unilinea). Asi se escribia un solo request en el estado de request si es que el servidor de origen no es pipeliner y se traicionaba a resposne, luego en response si se terminaba de recibir el response se volvía a request para que este le mande el siguiente request al origen. Así se procedía hasta que se hayan procesado todos los requests.

2.4. Append capa

Como nuestro servidor implementa pipelining incluso cuando el servidor no lo implementa nos pareció razonable que si el usuario pide capa a un servidor sin pipelining a través de nuestro proxy, el capa response contenga pipelining. Para esto primero probamos integrarlo dentro del estado de response directamente, el cual es el estado que se encarga del manejo inicial de la respuesta. Esto resultaba complejo y oscurecía el código ya que había un caso especial de response en donde los handlers de response hacían cosas distintas. Por esto decidimos implementar una solución parecida a las transformaciones en donde desde response se delega al estado append capa para que este maneje todo el capa request y luego vuelva a response. Así el estado de response se encarga de delegar a otros estados la transformación de los mensajes.

2.5. Parsing atado a los handlers de read.

Inicialmente se hacía el parsing de lo que se leía de los sockets en los handlers de read, pero esto traia problemas si el servidor origen no soportaba pipelining, y en principio siempre que hubiera datos leidos que quedaban en un buffer pero no eran parseados. Esto es, si un response

finalizaba por ejemplo y se debia pasar a request por no tener el servidor origen la opcion de pipelining, responses quedaban en el buffer pero no eran parseados. Como pasaba esto y el read era el encargado de parsear se debia esperar a una nueva escritura del origen. Optamos por eso en cambiar la logica del parseo al write y mantener el read solamente para la lectura.

2.6. Escapado y des escapado al transform

Inicialmente las respuestas iban a la etapa de transformación tal y como venían del servidor orígen. Esto si bien funcionaba con programas tales como cat traía problemas al momento de suponer que el programa de transformación no debería saber de POP3 y por ende no debería seguir las reglas del protocolo. Por esta razón se debió implementar parsers de escapado y des escapado y buffers para los mismos con el fin de que se de escape la respuesta del servidor origen antes de entrar a la transformación y luego se re escape para que lo reciba el cliente que si se sabe que debe entender de POP3

2.7. Resolución de nombres no bloqueante

Para resolver nombres de forma tal que no se bloquee el proxy se debieron utilizar threads aparte para realizar esta tarea y así, en el caso de bloquearse la resolución, el proxy puede seguir atendiendo pedidos.

2.8 Cantidad de buffers

Inicialmente el TPE seguía una estrategia muy similar a el server SocksV5 provisto por la cátedra, siempre se parseaba de un buffer a otro sin importar que el parsing no cambie nada del buffer que se lee al que se escribe. Esto termina causando que necesitemos una cantidad excesiva

de buffers ya que por ejemplo se necesitaban dos buffers para el request cuando perfectamente se podía usar uno. Por esta razón se modificó la estructura de parsers. Se le agregó a los buffers un puntero de parsing sobre el cual se va llevando el parseo, así en el caso del request se lee y se escribe del mismo buffer manejando lógica de punteros. De esta forma se redujo la cantidad de buffers y se evitaron problemas en cuanto a la ubicación de la data a enviar.

3. Limitaciones

3.1. Modificación del tamaño de los buffers

Al modificarse los buffers en tiempo de ejecución el proxy no reemplaza los buffers de las conexiones existentes por otro. Esto se podría hacer pero con gran cuidado ya que los buffers se están utilizando. En esta versión del TPE los buffers reflejan el nuevo tamaño al aceptar una conexión entrante.

3.2. Modificación de content-transfer-encoding

Al filtrar, además de poner el mensaje de reemplazo, como queremos que este se lea hay que modificar el valor de los headers Content-Type y Content-Transfer-Encoding a text/plain y quoted-printable respectivamente. De ésta manera nos aseguramos que el cliente pueda leer correctamente el mensaje. Sin embargo, no tuvimos en cuenta el caso en que el header Content-Transfer-Encoding se encontrara por sobre el de Content-Type. Si esto pasara, el programa funcionará correctamente con la excepción de que el encoding no se modificaría, pudiendo resultar en la ilegibilidad del mensaje.

3.3. Credenciales de los usuarios en el código

En esta versión del proxy este implementa los nombres de usuarios directamente sobre el código Esto resulta inflexible al momento de agregar, modificar o quitar usuarios administradores ya que para esto se necesita bajar el servidor, cambiar el código fuente, compilarlo y volverlo a subir

3.4. Pipelining inutilizado

Según el RFC1939 [POP3] un servidor puede implementar pipelining pero no capa. Para la detección de pipelining el proxy emite un CAPA request al servidor origen y si este contiene el string pipelining dentro de sus capacidades entonces sabe que es un servidor origen pipeliner. Como ni CAPA ni pipelining son obligatorios para los servidores POP3 se puede dar el caso, aunque raro, que un servidor implemente pipelining sin implementar CAPA. En este caso el proxy no detectara que el servidor origen efectivamente implementa pipelining y continuará mandando un request a la vez como si fuese no pipeliner.

3.4. Handling de la terminación

El proxy no implementa una terminación elegante con la desconección de los clientes, por esta razón no se puede mantener un trackeo limpio de las conexiones concurrentes ya que si se terminan por parte del cliente entonces no se registrará de manera apropiada.

4. Posibles extensiones

4.1 Agregado de Timeouts

Resultaría útil terminar una conexión que permaneció inactiva por más de una determinada cantidad de tiempo, así se podría mejorar la performance del proxy ya que se irían purgando las conexiones inactivas.

4.2 Repositorio de usuarios SPCP

Para poder remover las credenciales de los usuarios administradores del código debemos poder guardarlas de alguna otra manera. Una posible implementación más elegante es bajar los usuarios a un archivo de donde se levanten en tiempo de ejecución, esto permite que la modificación de los usuarios administradores sea tan simple como modificar un archivo y no se tenga que compilar el binario

4.3 Logging a disco

El logging en este momento se hace a consola, una posible mejora sería poder armar archivos de logging en donde se depositen los logs cada vez que se ejecuta el proxy. Así se puede tener trazabilidad de lo que pasó en el proxy incluso si se produce un reinicio o algo por el estilo.

4.4 Estadísticas no volátiles

Se podrían escribir las estadísticas a disco, en este caso se podría no solo soportar conexiones históricas desde que el proxy se empezó a ejecutar, sino que también conexiones históricas desde

el momento en que primero se ejecutó. Además no se perderían las estadísticas en el caso de un fallo lo cual resultaría ventajoso para una administrador.

4.5 Soporte de SSL y TLS

En esta versión del trabajo práctico no se soporta ni TLS ni SSL por lo que nada viaja encriptado sobre la red. Esto es particularmente peligroso en el caso de usuarios y contraseñas donde cualquiera que se encuentre sniffeando la red podrá ver los usuarios y las contraseñas que se intercambian en el proxy.

.

5. Conclusiones

El trabajo realizado representó un gran desafío para el equipo y logró poner en práctica conceptos esenciales de los protocolos de comunicación y el ambiente de programación unix que los rodea. Se pudo diseñar e implementar un protocolo cliente servidor mediante el protocolo de transporte SCTP así ejercitando los conocimientos no solo sobre el diseño de protocolos de aplicación sino que también la utilización provechosa del protocolo de transporte SCTP al momento de transportar los mensajes.

Se logró implementar un proxy POP3 el cual solamente conociendo reglas del protocolo permite transformar mensajes y aumentar las capacidades del servidor origen. El proxy además representó el desafío de la programación no bloqueante en el entorno UNIX, permitió al equipo enfrentarse con problemáticas de la programación orientada a eventos ya que se implementó el proxy con selectores que manejan máquinas de estado en base a señales de read o write.

Finalmente nos gustaría decir que el equipo siente que el trabajo práctico podría haberse logrado con mayor calidad dada una mejor organización del equipo. A pesar de esto se logró un proxy, un servidor SPCP, un cliente SPCP y un censurador de MIME types que cumplen con todas las funcionalidades estipuladas en menor o mayor medida.

6. Ejemplos de prueba

6.1. Se recibe un mail con MIMEs de tipo image/jpeg e image/png cuando se pedia filtrar image/*, el mail original (con imagenes acortadas a pocas lineas para mejor legibilidad) filtra todas las imágenes:

Original:

```
Date: Sun, 5 Nov 2017 18:59:10 -0300
From: juan <juan@juan-virtual-machine>
To: juan
Subject: ii
Message-ID: <20171105185910.5bec39ca@juan-virtual-machine>
Mime-Version: 1.0
Content-Type: multipart/mixed; boundary="MP_/XAkM/CBycDJWEUGbQ_LLaw3"

--MP_/XAkM/CBycDJWEUGbQ_LLaw3
Content-Type: text/plain; charset=US-ASCII
Content-Transfer-Encoding: 7bit
Content-Disposition: inline

Un png, un jpgeg
--MP_/XAkM/CBycDJWEUGbQ_LLaw3
Content-Type: image/png
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=favicon.png

iVBORw0KGgoAAAANSUhEUgAAACAAAAAgCAYAAABzenr0AAAAGXRFWHRTb2Z0d2FyZQBBZG9iZSBJ
r0wDx3aSBCVwujUIHQulwgvmAVsEObgHrLscTrZiiBgPIpeGkbYWItlIxJpJYx1J0lDJzxGO2zFr
SoikTdsxlfYa6cTn5KOu0XYenqwcvKS9JJePzmx9IPD8FXsIeAh4CPzvCfwjwAByxCObPlxpSQAA
AABJRU5ErkJggg==

--MP_/XAkM/CBycDJWEUGbQ_LLaw3
Content-Type: image/jpeg
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=favicon.jpg

/9j/4AAQSkZJRgABAQEASABIAAD/2wBDAAMCAgICAgMCAgIDAwMDBAYEBAQEBAgGBgUGCQgKCgkI
CQkKDA8MCgsOCwkJDRENDg8QEBEQCgwSExIQEw8QEBD/2wBDAQMDAwQDBAgEBAgQCwkLEBAQEBAQ
F5uEodq2EgvlBSApxSQpJ+yjzzzpo+jUBemaAwmzQfcKOjeatN4sNzZInF4i/opDVejJpXbLSP/Z

--MP_/XAkM/CBycDJWEUGbQ_LLaw3
Content-Type: text/plain
Content-Transfer-Encoding: 7bit
Content-Disposition: attachment; filename=hola.txt

hola mundo!

--MP_/XAkM/CBycDJWEUGbQ_LLaw3--
```

Diff:

```
--- ii_images.mbox  2018-11-06 12:08:07.885665500 -0300
+++ out 2018-11-06 12:08:33.342835800 -0300
@@ -13,25 +13,18 @@

 Un png, un jpgeg
 --MP_/XAkM/CBycDJWEUGbQ_LLaw3
-Content-Type: image/png
-Content-Transfer-Encoding: base64
+Content-Type: text/plain
+Content-Transfer-Encoding: 7bit
 Content-Disposition: attachment; filename=favicon.png
-
-iVBORw0KGgoAAAANSUhEUgAAACAAAAAgCAYAAABzenr0AAAAGXRFWHRTb2Z0d2FyZQBBZG9iZSBJ
-r0wDx3aSBCVwujUIHQulwgvmAVsEObgHrLscTrZiiBgPIpeGkbYWItlIxJpJYx1J0lDJzxGO2zFr
-SoikTdsxlfYa6cTn5KOu0XYenqwcvKS9JJePzmx9IPD8FXsIeAh4CPzvCfwjwAByxCObPlxpSQAA
-AABJRU5ErkJggg==
-
---MP_/XAkM/CBycDJWEUGbQ_LLaw3
-Content-Type: image/jpeg
-Content-Transfer-Encoding: base64
+
+
+Parte reemplazada.
+--MP_/XAkM/CBycDJWEUGbQ_LLaw3
+Content-Type: text/plain
+Content-Transfer-Encoding: 7bit
 Content-Disposition: attachment; filename=favicon.jpg
-
-/9j/4AAQSkZJRgABAQEASABIAAD/2wBDAAMCAgICAgMCAgIDAwMDBAYEBAQEBAgGBgUGCQgKCgkI
-CQkKDA8MCgsOCwkJDRENDg8QEBEQCgwSExIQEw8QEBD/2wBDAQMDAwQDBAgEBAgQCwkLEBAQEBAQ
-F5uEodq2EgvlBSApxSQpJ+yjzzzpo+jUBemaAwmzQfcKOjeatN4sNzZInF4i/opDVejJpXbLSP/Z
-
---MP_/XAkM/CBycDJWEUGbQ_LLaw3
+
+Parte reemplazada.
+--MP_/XAkM/CBycDJWEUGbQ_LLaw3
 Content-Type: text/plain
 Content-Transfer-Encoding: 7bit
 Content-Disposition: attachment; filename=hola.txt
```

7. Guia de instalación

Primero se debe clonar el repositorio u obtener una copia del mismo, puede clonarse mediante HTTP con el siguiente comando:

git clone https://${USER}@bitbucket.org/itba/pc-2018b-07.git

donde ${USER} es el usuario con acceso al repositorio.

Todos los elementos del trabajo práctico contienen un makefile autogenerado a través de CMake. Cada elemento del TPE se construye de la siguiente manera.

7.2. Instalación del proxy

El proxy contiene tanto el proxy pop3 propiamente dicho como el servidor SPCP utilizado para configuración y monitoreo. Para instalarlo se debe acceder a la carpeta /server/src y correr make. Una vez instalado se puede proceder a su ejecución.

7.2. Instalación del cliente SPCP

Para instalar el cliente dirigirse a /client/src y correr make.

7.3. Instalación del censurador de MIME types

para instalar el censurador de MIME types dirigirse a /stripmime y ejecutar make.

8. Instrucciones para la configuración

La configuración del proxy se puede hacer de dos maneras, en tiempo de ejecución mediante el cliente SPCP provisto en el TPE, o al momento de correrlo mediante las opciones posix que se proveen.

8.1. Configuración en tiempo de ejecución

Para configurar el proxy en tiempo de ejecución se debe inicializar el cliente, luego autenticarse mediante el usuario admin y contraseña admin. Finalmente se presenta el menú de configuración y monitoreo en donde se pueden modificar las transformaciones y los tamaños de los buffers.

8.2. Configuración mediante las opciones posix

El proxy acepta como argumentos múltiples opciones estilo posix para su configuración. Estas son:

- *-e filter-error-file* especifica a dónde se dirige la salida de error al ejecutar los filtros, por defecto es /dev/null.
- *-h* Imprime el diálogo de ayuda y termina.
- *-l pop3-address* especifica la dirección en la cual el proxy escuchara, por default son todas las interfaces.
- *-L config-address* especifica dónde estará escuchando el servicio SPCP, por default escucha en loopback.
- *-m message* el mensaje que dejara el filtro cuando censura.

- *-M censored-media-types* lista de media types que serán censurados
- *-o management-port* Puerto SCTP donde escuchar ael servicio SPCP, por default es el 9090
- *-p local-port* puerto donde escuchará el proxy para conexiones TCP entrantes, por default es el 1110
- *-P origin-port* puerto TCP donde estará escuchando el servidor de origen, por default es el 110
- *-t cmd* especifica el comando que se ejecutara para las transformaciones
- *-v* imprime la versión del proxy

8.3. Configuración del cliente SPCP

el cliente por default asume que el servidor spcp se encuentra en 127.0.0.1:9090, pero posee opciones de configuración para cambiar esto. Las opciones son estilo posix y son las siguientes:

- *-L config-address* dirección donde se encuentra el servidor SPCP
- *-o management-port* el puerto en donde se encuentra escuchando el servidor SPCP

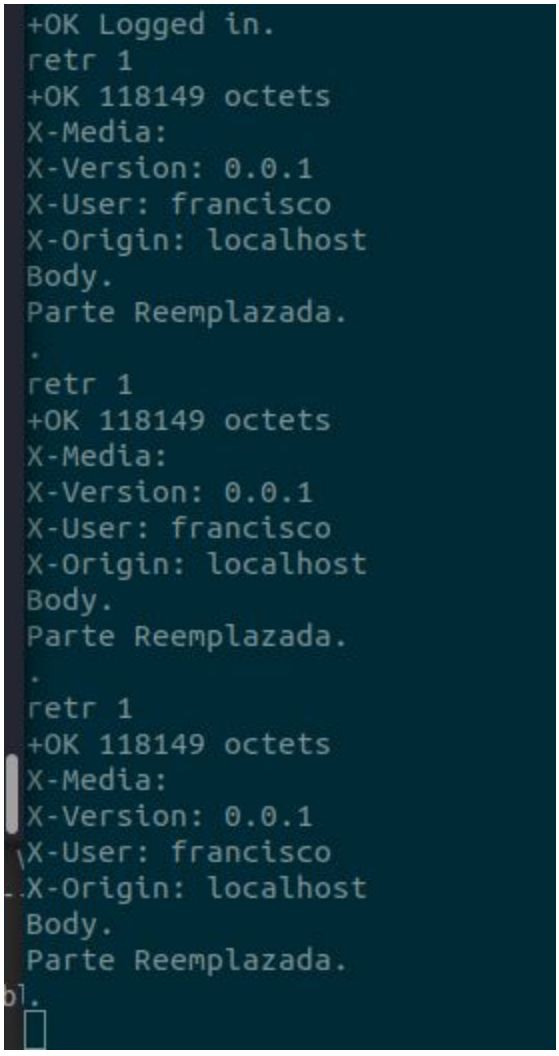
8.4. Configuración del censurador de MIME types

el censurador de media types tiene dos posibles entradas, stdin en cuyo caso se corre sin opciones, o un archivo en cuyo caso se corre como ‘./stripmime <archivo>’

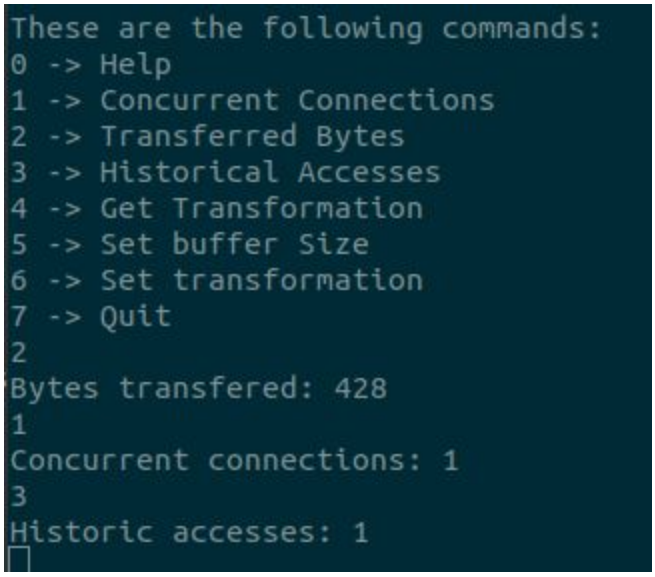
9. Ejemplos de configuración y monitoreo

9.1. Monitoreo del proxy

Primero abrimos la terminal A en la cual nos conectamos al proxy y ejecutamos un par de comandos.



Luego en la terminal B abrimos el cliente SPCP y miramos algunas métricas.



Como podemos ver nos marca correctamente que desde el inicio del proxy hubo 1 sola conexión, que hay una sola conexión concurrente y que se transfirieron 428 bytes por el proxy.

9.2. Configuración y monitoreo de las transformaciones

Este ejemplo demostrará el cambio de las transformaciones en tiempo de ejecución del servidor. En la terminal del cliente SPCP (A) primero nos logueamos como administrador y luego verificamos que se encuentra activa la transformación envs.sh.

```
francisco@Shakuras:~/itba/2c2018/protos/pc-2018b-07/client/src$ ./spcpClient
 Trying to connect to 127.0.0.1:9090
SPCP PROTOCOL CLIENT STARTED
        Please login
 Press 1 to login
1
Enter username
admin
Enter Password
admin
Login successful

These are the following commands:
0 -> Help
1 -> Concurrent Connections
2 -> Transferred Bytes
3 -> Historical Accesses
4 -> Get Transformation
5 -> Set buffer Size
6 -> Set transformation
7 -> Quit
4
Active transformation: bash envs.sh
```

Luego verificamos que se transforma efectivamente en la terminal B

```
retr 1
+OK 118149 octets
X-Media:
X-Version: 0.0.1
X-User: francisco
X-Origin: localhost
Body.
Parte Reemplazada.
.
```

Ahora regresamos a la terminal A y modificamos la transformación ingresando 6 para cambiar la transformación y luego ingresando la transformación.

```
6
 Enter new transformation command
bash byte_stuffed.sh
Transformation set
```

finalmente ejecutamos retr 1 de nuevo y vemos que la transformación cambio.

```
retr 1
+OK 118149 octets
X-Header: true
..hola
..
.
```

9.3 Modificación del tamaño del buffer

como vemos en la terminal se puede modificar el tamaño de los buffers del proxy mediante el cliente SPCP.

```
francisco@Shakuras:~/itba/2c2018/protos/pc-2018b-07/client/src$ ./spcpClient
 Trying to connect to 127.0.0.1:9090
SPCP PROTOCOL CLIENT STARTED
        Please login
 Press 1 to login
1
Enter username
admin
Enter Password
admin
Login successful

These are the following commands:
0 -> Help
1 -> Concurrent Connections
2 -> Transferred Bytes
3 -> Historical Accesses
4 -> Get Transformation
5 -> Set buffer Size
6 -> Set transformation
7 -> Quit
5
 Enter new buffer size (in decimal)
1234
New Buffer size set
```