

72.07 Protocolos de comunicación

2° Cuatrimestre del 2021

Trabajo Práctico Especial

“Servidor Proxy POP3”



Grupo 6

Integrantes del grupo

Oriana, Juan Pablo - 60621

García Montagner, Santiago - 60352

Cerdeira, Tomás - 60051

Catolino, Lucas - 61817

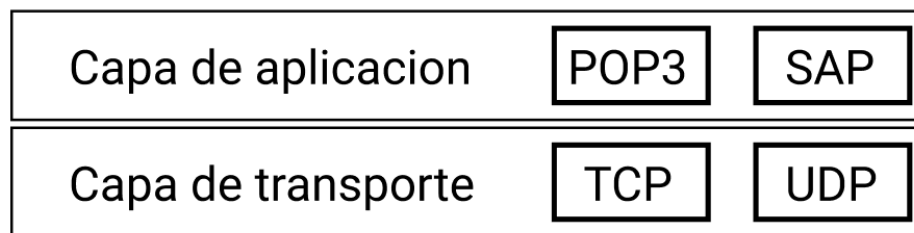
Fecha de entrega: 23/11/2021

Índice

1. Protocolos usados	1
1.1. De aplicación	1
Post Office Protocol - Version 3 (POP3)	1
Server Admin Protocol - Version 1 (SAP)	1
1.2. De transporte	1
Transmission Control Protocol (TCP)	1
User Datagram Protocol (UDP)	2
2. Server Admin Protocol (SAP)	3
2.1. Introducción	3
2.2. Estructura de un paquete SAP	3
2.2.1. Datagrama de un REQUEST	3
2.2.2. Datagrama de un RESPONSE	6
3. Aplicaciones desarrolladas	9
3.1. Main	9
3.2. Proxy POP3	9
3.2.1. Selector y pool de conexiones	9
3.2.2. Estados de conexión	10
3.2.3. Parsers	11
3.3. Server y Cliente SAP	12
3.4. Logging	12
4. Problemas encontrados	13
5. Limitaciones	13
6. Posibles extensiones	14
7. Conclusiones	14
8. Ejemplos	15
8.1. LIST	15
8.2. RETR	15
8.3. DELE y RETR	15
8.4. DELE y REST	16
8.5. DELE y LIST	16
8.6. STAT	16
8.7. NOOP	16
9. Guía de instalación	17
10. Instrucciones de configuración	17
11. Ejemplos de configuración	19
11.1. Seteo de filtros	19
11.2. Consulta de filter	20

11.3. Uso de current y historic	20
11.4. Uso de bytes	21
11.5. Redirección de errores	21
11.6. Cambio de timeout	22
12. Diseño del proyecto	23
13. Casos de prueba	24
13.0. Convenciones	24
13.1. Argumentos de las aplicaciones	25
13.1.0. Binding en puertos default correcto	25
13.1.1. Cambio de puertos e interfaces default	25
13.2. Disponibilidad	26
13.2.0. Origin server (IPV4) no presta servicio	26
13.2.1. Origin server (IPV6) no presta servicio	26
13.2.2. Origin server con múltiples direcciones IP (una falla)	26
13.2.3. Comportamiento origin server resuelve DNS IPV6	28
13.2.4. Proxy Server como Origin Server (recursividad)	29
13.2.5. Performance e inmutabilidad de bytes	30
13.2.6. Concurrencia	31
13.2.7. Desconexión repentina de clientes	32
13.2.8. Desconexión repentina del origin server esperando comando	33
13.2.9. Desconexión repentina del origin server durante un RETR	34
13.2.10. Lecturas parciales desde el cliente	35
13.3. Uso del proxy POP3	37
13.3.1. Comportamiento origin server que da servicio comandos básicos	37
13.3.2. Pipelining cliente	39
13.3.3. Trailing spaces no afectan a los comandos	39
13.4. Pipelining	41
13.4.1. Pipelining con origin server que no soporta pipelining (de forma declarada)	41
13.4.2. Pipelining hacia origin server (declara que lo soporta)	41
13.5. Integración de la Transformaciones de mensajes	42
13.5.1. Transformación utilizando cat	42
13.5.2. Seteo de variables de entorno	43
13.5.3. Respuesta lazy	43
13.5.4. Byte Stuffed	44
Anexo - Mensaje “Grande”	47

1. Protocolos usados



1.1. De aplicación

- Post Office Protocol - Version 3 (POP3)
 - [RFC - 1939](#)

Como el principal objetivo de este proyecto es llevar a cabo un proxy que atienda y responda pedidos entrantes a un servidor POP3, se tuvo en cuenta el protocolo implementado por el mismo. Desde los indicadores de estados “+OK” y “-ERR”, hasta la lista de comandos básicos disponibles, fueron extraídos y analizados de la documentación original para su correcta implementación en el proxy.

- Server Admin Protocolo - Version 1 (SAP)
 - [Sección SAP](#)

Una de las funcionalidades que el proxy debe ofrecer es la posibilidad de implementar mecanismos que permitan recolectar métricas; cantidad de conexiones (históricas y concurrentes), cantidad de bytes transferidos y cualquier otro dato relevante para su análisis. Además, distintas configuraciones del proxy como tamaños de buffers, tiempo definido para un “timeout” y cambios de las transformaciones llevadas a cabo por él, entre otras, deben poder ser obtenidas y modificadas en tiempo de ejecución sin necesidad de reiniciar el servidor.

Para todo lo mencionado anteriormente, se decidió desarrollar e implementar el protocolo SAP.

1.2. De transporte

- Transmission Control Protocol (TCP)
 - [RFC - 793](#)

Como está definido en el RFC de POP3, el mismo utiliza el protocolo de transporte TCP. Comienza su servicio escuchando conexiones en el puerto 110, una vez que un cliente se quiere conectar, se establece una conexión entre él y el host.

Como los tamaños de los datagramas en este protocolo NO son fijos y la totalidad de un mensaje puede venir repartida entre varios paquetes, se tuvo que llevar a cabo un manejo para procesar y efectuar los comandos deseados de forma no bloqueante.

- User Datagram Protocol (UDP)

- [RFC - 768](#)

Para simplificar su implementación y teniendo en cuenta que no tiene mucho sentido mantener una sesión para el servicio que ofrece, se decidió usar UDP como protocolo de transporte para SAP.

En este caso, que los tamaños de los datagramas sean fijos y los datos lleguen de forma directa, no separada entre varios paquetes, redujo la complejidad de su implementación significativamente.

Esta decisión de desarrollo se tomó teniendo en cuenta lo llevado a cabo en el TPE1 de la materia. En el mismo, se pidió implementar un servidor que recibiera datagramas UDP cambiando configuraciones del mismo, como su localización. Además, se podían pedir estadísticas como cantidad de conexiones realizadas desde que inició la ejecución, parecido a lo pedido para el TPE2.

En situaciones en las que se desea obtener una respuesta simple y rápida, como esta, UDP funciona mejor. En general, el hecho de que la respuesta esté en un único paquete, simplifica su implementación y la posibilidad de implementar un protocolo propio, como el SAP.

2. Server Admin Protocol (SAP)

2.1. Introducción

La siguiente sección describe el protocolo SAP y su correcta implementación para llevar a cabo un cliente que interactúe con el servicio de management del proxy.

La orientación del protocolo SAP no es proporcionar amplias operaciones de administración sobre el proxy; simplemente la obtención de métricas relevantes que ayuden a monitorear la operación del sistema y la posibilidad de contar con mecanismos que cambian configuraciones del servidor en tiempo de ejecución.

El protocolo de configuración utiliza UDP como protocolo de transporte. Para su uso, se requiere estar autenticado mediante un id propio del servidor. En cada request, el mismo debe ser incluido en el datagrama para asegurar que la persona intentando obtener información o modificar comportamiento esté capacitada y habilitada para hacerlo.

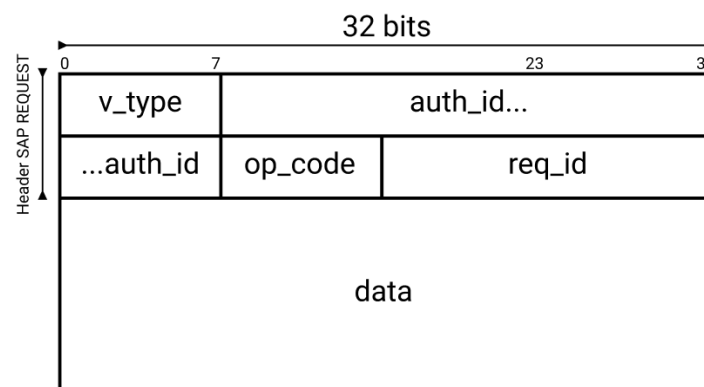
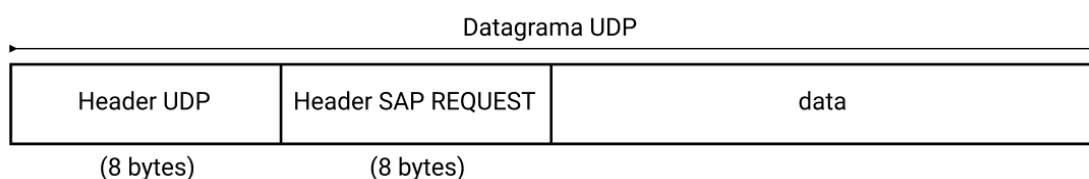
Los request tienen un header de 8 bytes mientras que los response tienen uno de 5 bytes. A continuación sigue una sección de datos que queda determinada por el tipo de operación que se está ejecutando.

2.2. Estructura de un paquete SAP

Los paquetes UDP tienen un tamaño fijo que varía entre 8 y 65.535 bytes. En ellos, además de incluir el header UDP definido en su protocolo, incluyen, normalmente, el header del protocolo de aplicación que están transportando además de los datos.

En este caso, el paquete UDP incluye el header SAP correspondiente a un request o response y la data a enviar.

2.2.1. Datagrama de un REQUEST



Descripción de los campos

→ **v_type:** Versión del protocolo usada

◆ Tipo de dato: uint8_t (1 byte)

Valor	Tipo	Descripción
1	SAP_V_1_0_0	Usar la versión 1.0.0 del SAP

→ **auth_id:** Autenticador de “sesión”

◆ Tipo de dato: unsigned int de 4 bytes big endian.

◆ Usado para que solo administradores que cuenten con las credenciales necesarias para autenticarse puedan usar el servicio

→ **op_code:** Código del comando a llevar a cabo

◆ Tipo de dato: unsigned int de 1 byte

Valor	Tipo	Operación	Data
0	OP_STATS	Obtener las estadísticas del proxy	1 byte que determina el tipo de stat que se requiere (0 conexiones históricas, 1 conexiones actuales, 2 bytes transferidos)
1	OP_GET_BUFF_SIZE	Obtener el tamaño del buffer	-
2	OP_SET_BUFF_SIZE	Modificar el tamaño del buffer	2 bytes con el tamaño (unsigned)
3	OP_GET_TIMEOUT	Obtener el tiempo definido para un timeout	-
4	OP_SET_TIMEOUT	Modificar el tiempo definido para un timeout	1 byte con el timeout (unsigned)
5	OP_GET_ERROR_FILE	Obtener el archivo definido para la salida de errores	-
6	OP_SET_ERROR_FILE	Modificar el archivo definido para la salida de errores	String null terminated con el error file
7	OP_GET_FILTER	Obtener el filtro aplicado en las	-

		transformaciones	
8	OP_SET_FILTER	Modificar el filtro aplicado en las transformaciones	String null terminated con el error file
9	OP_IS_FILTER_WORKING	Informa si el filtro esta encendido	-
10	OP_TOGGLE_FILTER	Cambia el estado de encendido del filtro	Unsigned byte. 0 apaga y 1 prende.

→ **req_id:** Identificador del request

- ◆ Tipo de dato: unsigned int de 2 bytes big endian
- ◆ El response que corresponda al pedido va a tener que coincidir en este campo.

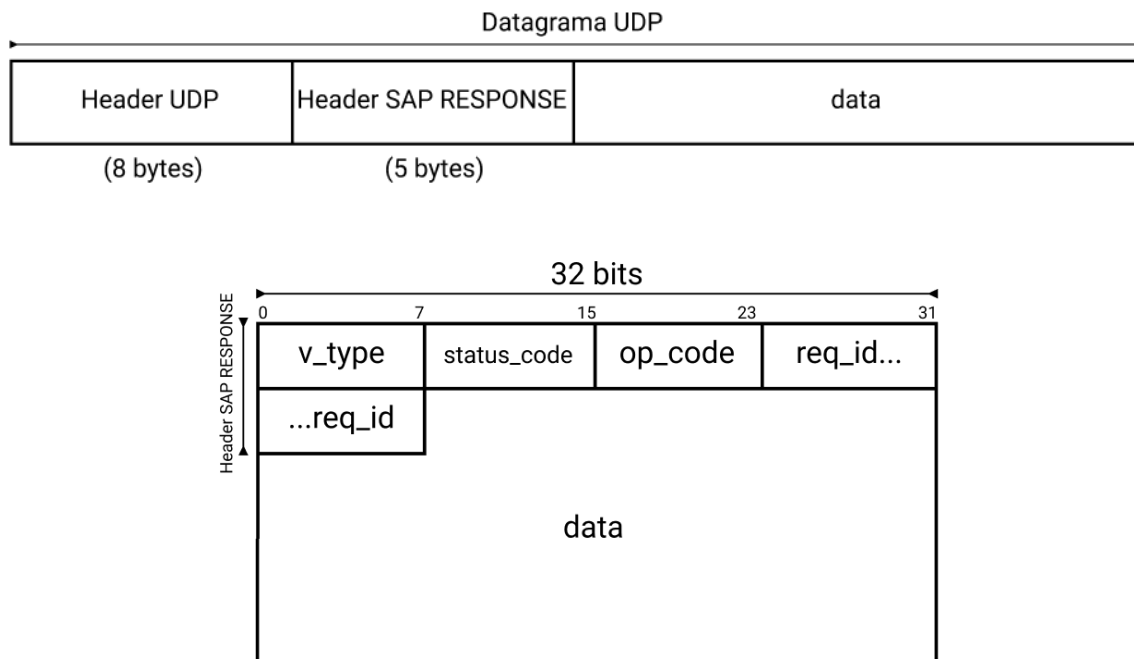
→ **data:** La data de un request queda estrictamente determinada por el tipo de operación que se realiza (ver arriba). Los tipos de datos contemplados son:

- ◆ Singles (unsigned int de 8 bits)
- ◆ Shorts (unsigned int de 16 bits)
- ◆ Longs (unsigned int de 32 bits)

(Todos codificados en big endian)

- ◆ Strings null terminated
- ◆ Por supuesto puede llegar a haber request sin data (los getters, por ejemplo).

2.2.2. Datagrama de un RESPONSE

Descripción de los campos

→ **v_type:** Versión del protocolo usada

◆ Tipo de dato: unsigned int de 1 byte

Valor	Tipo	Descripción
1	SAP_V_1_0_0	Usar la versión 1.0.0 del SAP

→ **status_code:** Código de respuesta del servidor

◆ Tipo de dato: unsigned int de 1 byte

Valor	Tipo	Descripción
0	SC_OK	Comando efectuado correctamente
10	SC_COMMAND_UNSUPPORTED	Comando no admitido
11	SC_COMMAND_INVALID_ARGS	Argumentos inválidos
12	SC_NO_FILTER	No hay filter en el proxy para una operación que requiere de tener uno
20	SC_UNAUTHORIZED	No autorizado
30	SC_VERSION_UNKNOWN	No se reconoce la versión
40	SC_INTERNAL_SERVER_ERROR	Error interno del servidor

→ **op_code:** Código del comando elegido para llevar a cabo

◆ Tipo de dato: unsigned int de 1 byte

Valor	Tipo	Operación	Data
0	OP_STATS	Obtener las estadísticas del proxy	La respuesta del stat pedido en 4 bytes unsigned big endian.
1	OP_GET_BUFF_SIZE	Obtener el tamaño del buffer	2 bytes con el tamaño (unsigned)
2	OP_SET_BUFF_SIZE	Modificar el tamaño del buffer	-
3	OP_GET_TIMEOUT	Obtener el tiempo definido para un timeout	1 byte con el timeout (unsigned)
4	OP_SET_TIMEOUT	Modificar el tiempo definido para un timeout	-
5	OP_GET_ERROR_FILE	Obtener el archivo definido para la salida de errores	String null terminated con el error file
6	OP_SET_ERROR_FILE	Modificar el archivo definido para la salida de errores	-
7	OP_GET_FILTER	Obtener el filtro aplicado en las transformaciones	String null terminated con el error file
8	OP_SET_FILTER	Modificar el filtro aplicado en las transformaciones	-
9	OP_IS_FILTER_WORKING	Informa si el filtro esta encendido	0 si esta apagado y 1 si esta prendido.
10	OP_TOGGLE_FILTER	Cambia el estado de encendido del filtro	-

OBS: la respuesta incluye el campo **op_code** para simplificar el manejo de la respuesta teniendo en cuenta sus posibles tamaños.

→ **req_id:** Identificador del request

◆ Tipo de dato: unsigned int de 2 bytes big endian

◆ Debe coincidir con el request al cual se está respondiendo.

→ **data:** La data de un response queda estrictamente determinada por el tipo de operación que se realiza (ver arriba). Los tipos de datos contemplados son:

◆ Singles (unsigned int de 8 bits)

◆ Shorts (unsigned int de 16 bits)

◆ Longs (unsigned int de 32 bits)

(Todos codificados en big endian)

- ◆ Strings null terminated
- ◆ Por supuesto puede llegar a haber request sin data (los getters, por ejemplo).

3. Aplicaciones desarrolladas

3.1. Main

El main se ocupa de, luego de parsear adecuadamente las opciones recibidas por argumento, levantar todos los sockets pasivos para la ejecución del proxy. Para eso, se intenta disponer **siempre** tanto de un socket pasivo IPv4 como un IPv6 para las conexiones con el proxy y manager. Estas funciones puede que fallen (por ejemplo si se pasa como parámetro una IPv4, al intentar levantar IPv6 va a fallar) pero ***jamás puede suceder que no se logre establecer al menos un socket pasivo para el proxy y uno para el manager.*** Esa es condición necesaria para que el main aborte la ejecución y el programa termine.

Una vez llevado a cabo lo anterior, se setean los timeouts y los handlers para los sockets pasivos, se los subscribe al selector y comienzan las rutinas de atención correspondientes.

3.2. Proxy POP3

Servidor concurrente no bloqueante que consiste de los siguientes módulos:

- Selector
- Estructura de una conexión
- Pool
- Estados de conexión / Máquina de estados (STM)
- Parsers
- Filtros

3.2.1. Selector y pool de conexiones

El manejo de intereses de los file descriptors de la aplicación fue manejado gracias al código fuente **selector** provisto por la cátedra.

En base al estado de la conexión, el selector permite suscribir alguno de los files descriptors de la estructura a cierto interés. Estos “intereses” pueden ser: leer, escribir o ninguno de los anteriores. Por otro lado, a cada file descriptor se le asigna un handler encargado de manejar y coordinar las acciones a llevar a cabo tras recibir o enviar bytes de la comunicación. Estos handlers varían según el estado actual de la conexión.

Luego de cada read o write, los intereses deben ser recalculados con una función **compute_interests** que suscribió el fd o no dependiendo de chequeos específicos a cada etapa de la conexión.

A cuáles intereses se le presta atención, así como cuáles son los criterios para que un fd se vuelva a suscribir o no a cierto interés, es algo que queda completamente determinado por el estado de la conexión donde se encuentre (ver [inciso 3.2.2](#)).

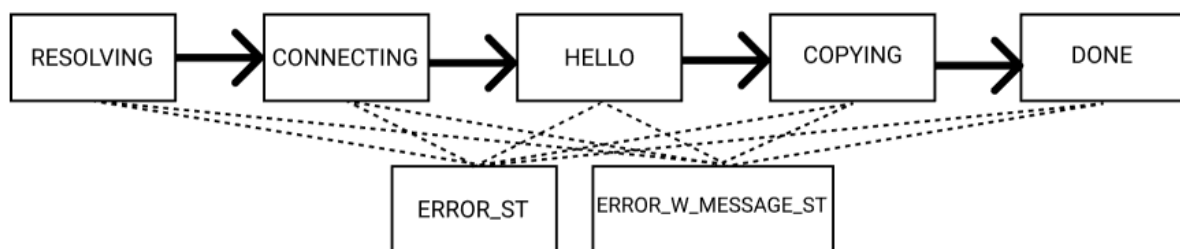
Para manejar más fácilmente el estado de cada cliente, cada conexión fue modelada como una estructura de datos. La misma, se encarga de manejar todo lo relacionado a una conexión: su creación, inicialización, atención y destrucción.

Al recibir una conexión en uno de los sockets pasivos, sea IPv4 o IPv6, tras resolver los dominios y establecer la conexión entre el servidor de origen POP3 y el cliente, se cuenta con una estructura de tipo **connection**. Esta, contiene toda la información relevante y necesaria para manejar dicha conexión: file descriptors del origen y cliente, buffers de escritura y lectura, buffers para la escritura en el filter, los parsers para cada etapa de la conexión, las representaciones adecuadas de las direcciones donde nos tenemos que conectar, entre otros. Todos los buffers se crean, por defecto, con un tamaño de 2048 bytes.

Las conexiones llevan cuenta de las referencias establecidas, sabiendo así cuando corresponde su “destrucción”. Aun así, no se eliminan directamente, van a parar a un pool de conexiones destruidas donde su estructura puede volver a ser utilizada en un futuro sin necesidad de instanciar una nueva estructura en memoria. De esta forma además, se ahorra tiempo de liberación de recursos ya que no se liberan en el instante que una conexión es terminada.

De todas formas, las conexiones inactivas no viven permanentemente. Estas están regidas por un valor de **TIMEOUT** (alterable por el management), que determina después de qué punto de inactividad se mata la conexión (para evitar floodear el server).

3.2.2. Estados de conexión



El manejo de los estados es simple. Inicialmente se determina si hay que resolver el host del origen o no. De ser así se inicia en el estado **RESOLVING**, donde se abre un hilo de ejecución para resolver el listado de IPs que le corresponden. Una vez que termina ese hilo y el selector es avisado, se pasa a una instancia **CONNECTING**, donde se intenta conectar al listado de direcciones que se pudo resolver. Si al menos una conecta bien, se avanza.

También puede suceder que llegue un address que no debía pasar por ningún proceso de resolución. En este caso saltó directamente al estado **CONNECTING** desde un primer instante. En este caso, a diferencia de con un hostname, el resultado posible es binario: o bien se conecta y avanza, o no se puede conectar y no tiene con qué más probar.

Ya sea con resolución o no, si efectivamente se pudo conectar, teniendo así los 2 sockets correspondientes, comienza la espera del mensaje inicial por parte del servidor **HELLO**. Según el RFC, se espera que el servidor comience con un mensaje del estilo “+OK msg”. Si esto ocurre sin inconvenientes, se pasa al estado **COPYING**.

En el copying, se lee **comando a comando** lo que envía el cliente (el pipelining siempre es manejado internamente por el proxy, y no se envía de corrido al origen aunque el mismo pueda llegar a soportarlo). Constantemente se cuenta con una “ida y vuelta” de pregunta-respuesta entre el cliente y el servidor. Hay un par de casos especiales: de recibir un comando **CAPA** proveniente del cliente, el proxy se asegura de parsear y agregar el comando

PIPELINING como disponible, de no encontrarse en la respuesta del original, ya que siempre debemos avisar que lo soportamos. En el caso en el que se recibe un request de **RETR** y el origen responde exitosamente, se dan dos escenarios. El primero, es que no esté activado el filtro, aquí no se hace nada distinto a el resto de las operaciones, copiarlas en un buffer para que el cliente lo consuma. El segundo, el cliente quiere filtrar la respuesta; aquí el manejo es más complejo ya que se debe iniciar el filtro. El flujo es el siguiente:

- Se parsea la respuesta del origen y se analiza si es de interés para el filtro.
- Se inicia el proceso de transformación, el cual fue definido por el cliente, además se abren los pipes de comunicación.
- Una vez iniciado el nuevo proceso, comienza un ida y vuelta entre el proxy y el filtro. El proxy le envía solamente el cuerpo del email, quitando (con ayuda del `filter_parser`) la primera línea de éxito que envía el origen y los caracteres de escape que agrega el servidor POP3 a los CRLF del cuerpo del mensaje. Una vez que el filtro responde con la transformación realizada, se le agrega la primera línea del mensaje de éxito definida, escapando, de ser necesario, los CRLF para enviarle la respuesta al cliente en el formato que define el protocolo.
- Una vez parseada la respuesta del filtro, el proxy deja la respuesta en un buffer el cual el cliente consumirá.
- Cuando se termina de filtrar, el proxy recibe del proceso un EOF lo que indica que ya no hay más nada para hacer y se cierra el filtro, matando el proceso creado.

Por último, existen tres estados de los cuales no se retorna:

- **DONE**: la conexión finalizó de forma exitosa.
- **ERROR_W_MESSAGE**: ocurrió algún error y se lo hace llegar al cliente.
- **ERROR**: ocurrió un error que no permite ni hacer llegar al cliente lo ocurrido.

En cualquier momento de la ejecución y por distintos motivos se puede llegar a un estado de error. Es prioridad absoluta del proxy hacer lo posible para informar al cliente de este error. Sin embargo, es posible que ciertos errores (por ejemplo una suscripción del cliente en el selector) no permitan hacer ningún tipo de aviso.

3.2.3. Parsers

Todos los parsers utilizados en la implementación trabajan de una forma similar. En general, todos trabajan sobre las mismas bases fundamentales:

- Cálculo del tamaño de la línea (para asegurar de no leer demas)
- El estado CRLF (para saber cuándo llega el “\r\n” que representa el final del comando)
- El estado del parser en sí

Los dos parsers más interesantes para analizar son el de comandos y el de respuestas. El parser de comandos contrasta el comando actual con los posibles, definidos en el RFC,

acotando de esta manera su resolución. Una vez clasificado, se guarda si el mismo es, o no, multilínea (para saber qué esperar en la respuesta) y, de contar con, sus argumentos.

El parser de respuestas utiliza la información del comando leído para saber qué esperar y su comportamiento a continuación. Los tres pilares que analiza son:

- Si es monolínea o multilínea.
- Si es un **RETR**, por ende debe filtrar el cuerpo de la respuesta cuando pase la primera línea.
- Si es un **CAPA**, teniendo que agregar tal vez la disponibilidad de **PIPELINING** a la lista.

3.3. Server y Cliente SAP

Tanto para el manejo del cliente como el server SAP implementado, se usaron una serie de utilidades implementadas en **sap.c**. Uno de sus principales beneficios es la traducción de estructuras de datos en código, representativas de requests y responses, a forma de buffer o de forma de buffer, a dichas estructuras, simplificando su manejo y operaciones, trabajando eficientemente entre la abstracción de código y el bloque de chars del datagrama UDP. Estas funciones son lo suficientemente inteligentes para guardar en un tipo unión genérico **data**, el tipo de dato que corresponda al **op_code** recibido.

El servidor recibe y parsea los request entrantes, verificando que los mismos cumplan con todos los requisitos propuestos por el protocolo (versionado, autorización, **op_code** válido, argumento válido), retornando un error de no ser así. Si todas las condiciones se cumplen, se efectúa la operación deseada; con la estructura **pop3_proxy_state**, quien contiene el estado actual del proxy, se obtienen las estadísticas o se altera el estado correspondiente.

El cliente gestor, ofrece una serie de comandos case sensitive para llevar a cabo las operaciones sobre el proxy de manera rápida y ágil. Cuenta con 15 comandos que encapsulan todas las operaciones posibles definidas por el protocolo SAP. (ver la tabla de comandos de la sección [10. Instrucciones de configuración](#))

OBS: hay más comandos que **op_codes** ya que algunos comandos representan una combinación entre **op_code** y parámetro específico.

El token de autenticación con el cual se se debe verificar el cliente con el servidor, puede ser definido en una variable de estado llamada **SAP_AUTH** previo a la ejecución del servidor (esta variable no contempla modificaciones en ejecución).

3.4. Logging

El logging de las aplicaciones desarrolladas se maneja a cuatro niveles: INFO, DEBUG, ERROR y FATAL. En general se intentó no evadir ninguno de estos eventos loggeables:

- Errores que implican la detención de la ejecución

- Errores que alteran el flujo natural del programa
- Información de nuevas conexiones o nuevos usuarios en el proxy
- Información sobre el estado de cada conexión y en qué parte de la STM se encuentran.

4. Problemas encontrados

Algunos de los problemas encontrados durante el desarrollo del TPE fueron:

- En los parches había archivos con los que no se contaba (como un readme y un makefile). Se limpió el parche y funcionó.
- Durante el desarrollo de las aplicaciones, en particular del proxy, lo más difícil fue comprender el flujo del selector y la máquina de estados provistos por la cátedra. Esto representó una demora de varios días hasta que se los logró entender y manipular. Fue conveniente programar un proxy TCP antes de arrancar con el flujo de POP3 en sí. Una vez resueltos estos problemas de comprensión, el resto del trabajo se hizo mucho más ameno.
- Cómo hacer que la respuesta de RETR con filtro sea parseada de forma tal que la primera línea salga sin ser transformada y el resto del mensaje si. Se tuvo que reestructurar el parser de respuestas y hacer uno propio para el filtro, pero finalmente pudo ser resuelto.

5. Limitaciones

Por la naturaleza del selector que está detrás del selector, la cantidad de conexiones que se permiten están limitadas por la cantidad de fds que puede manejar este. El selector puede manejar hasta 1024 fds, de los cuales 2 ya están ocupados por STDOUT y STDERR y 4 potencialmente pueden llegar a estar ocupados por sockets pasivos. Esto nos deja 1018 fds libres y, considerando que cada conexión puede llegar a utilizar 4 fds (1 origen + 1 cliente + 2 filtro), la cantidad máxima de conexiones está claramente acotada por esta limitación técnica.

Por otro lado, como el protocolo POP3 usado para el desarrollo de este trabajo fue el [RFC - 1939](#), aplicaciones de Mail User Agents (MUAs) que implementan la última versión del mismo, [RFC - 5034](#), no son compatibles con este proxy. Esto tiene mucho que ver con una nueva forma de hacer el HELLO que no está contemplado en nuestros parsers.

Un ejemplo de esto es si se intenta configurar como servidor POP3 al proxy en el Thunderbird, MAU que viene por defecto en la mayoría de las distribuciones de Linux. El mismo, una vez que establece la conexión con el servidor, utiliza mecanismos de autenticación definidos en la extensión *Simple Authentication and Security Layer* (SASL) acoplada al protocolo POP3 en su última versión.

Mediante el comando **AUTH**, el MUA intenta autenticar al cliente comenzando una negociación SASL. De completar dicha negociación, el servidor responde un “+”.

El comportamiento descrito no es el esperado por el servidor proxy implementado, por lo que las máquinas de estado devuelven estados de error, cortando la ejecución del mismo e imposibilitando la autenticación del usuario.

OBS: Usando [Evolution](#) - cliente de mail que parecería usar la versión soportada de POP3 por el proxy, se puede trabajar tranquilamente.

Por último, de modificar el tamaño del buffer durante la ejecución del proxy, aquellas estructuras de conexiones destruidas que formen parte del pool, continuarán con el tamaño del buffer anterior y no el modificado.

6. Posibles extensiones

- Habilitar envío de comandos pipelineados al origen directamente si ya se sabe que el origen los habilita, en vez de dividirlos como se vino haciendo a lo largo del trabajo.
- Expandir las capacidades del manager en cuanto a lo que puede modificar (se podrían modificar dinámicamente puertos o addresses) y lo que puede informar (se podría hacer un informe más granular sobre la transferencia de bytes: de qué buffer viene cada uno, etc.)
- En el caso de que el filtro sea un comando invalido, en el proxy actual se decidió que se logee el error al error file designado y el RETR no va a poder codificar nada. Esto es distinto al caso donde falla la creación del proceso transform, donde se pasa a un copiado transparente. Se podría implementar que si el filtro es un comando invalido, se empiece a copiar transparentemente también.

7. Conclusiones

Durante este mes de programación, diseño y construcción, sentimos que realmente pudimos consolidar y confirmar gran parte de los conocimientos adquiridos durante el transcurso de la materia. Pudimos trabajar a fondo con un protocolo de comunicación fuertemente establecido como lo es el POP3 y por otro lado diseñar e implementar uno propio, inspirándonos en las distintas estrategias de construcción que vimos a lo largo del cuatrimestre. Una cosa es aprender e interiorizarse sobre todos los conceptos teóricos, y otra muy distinta es plasmarlos en el desarrollo de un trabajo práctico como el llevado a cabo.

Más allá de que fue un camino lleno de obstáculos y dificultades, sentimos que pudimos organizarnos adecuadamente como equipo de trabajo y llegamos óptimamente al momento de entrega con todo lo que nos habíamos propuesto.

8. Ejemplos

8.1. LIST

```
LIST
+OK 5 messages:
1 537
2 545 del cuatrimestre.
3 541a muy distinta es
4 540
5 694489434
. seguimos que pudimos
```

Con el comando LIST sin argumentos se recibe una línea por cada mail, con sus respectivos octetos.

8.2. RETR

```
RETR 1
+OK 537 octets
Return-Path: <lcatolino@itba.edu.ar>
X-Original-To: kali@kali.kali
Delivered-To: kali@kali.kali
Received: from kali.kali (localhost [127.0.0.1])
        by kali.kali (Postfix) with ESMTP id 99EE11616
39
        for <kali@kali.kali>; Wed, 10 Nov 2021 16:12:0
2 -0300 (-03)
From: <kali@kali.kali>
To: <kali@kali.kali>
Message-Id: <20211110191248.99EE1161639@kali.kali>
Date: Wed, 10 Nov 2021 16:12:02 -0300 (-03)

DATA
MIME-version: 1.0
Content-type: text/plain; charset= UTF-8
Content-transfer-Encoding: quoted-printable

Mail de prueba
.
```

Con el comando RETR msg se recibe el mensaje a leer (siendo msg el número de mensaje). El mensaje a leer no puede estar marcado como eliminado (como se ve en el ejemplo 8.1.3).

8.3. DELE y RETR

```
DELE 1
+OK Marked to be deleted.
RETR 1
-ERR Message is deleted.
```

Si se quiere leer un mensaje marcado como eliminado, se recibe un error.

8.4. DELE y REST

```
DELE 1
+OK Marked to be deleted.
DELE 1
-ERR Message is deleted.
RSET
+OK
DELE 1
+OK Marked to be deleted.
```

El comando DELE recibe el número de mensaje a eliminar (que no puede estar marcado como eliminado), y se lo marca como eliminado. No se lo elimina sino hasta después del UPDATE.

8.5. DELE y LIST

```
LIST 4
+OK 4 540
DELE 4
+OK Marked to be deleted.
LIST 4
-ERR Message is deleted.
```

Opcionalmente, el comando LIST puede recibir el número de un mensaje (que no puede estar marcado como eliminado). Si el mensaje existe, se responde una línea con la información de ese mensaje. Sino, se recibe un error.

8.6. STAT

```
LIST
+OK 5 messages:
1 537
2 545
3 541
4 540
5 694489434
.
STAT
+OK 5 694491597
█
```

Con el comando STATE, el servidor responde “+OK X Y” (X: cantidad de mensajes, Y: tamaño en octetos). En este ejemplo se puede ver que 694491597 corresponde a la suma de los octetos mostrados en LIST.

8.7. NOOP

```
NOOP
+OK
█
```

El comando NOOP corresponde a NO OPERATION. No hace nada, y el servidor responde +OK.

9. Guía de instalación

Para construir el proyecto, es tan fácil como hacer un make en el directorio principal:

```
foo@bar:~/Pop3Filter/$ make all
```

En ese momento se generarán dos (2) ejecutables. Uno en el directorio principal bajo el nombre **pop3filter**, que es el encargado de correr el proxy en sí, y otro dentro del directorio **manager_client** con el nombre **client**, que servirá como cliente del server de management.

10. Instrucciones de configuración

El cliente de configuración es muy sencillo de usar. Al momento de correrlo, lleva dos comandos obligatorios, el address y el puerto donde está escuchando el server management, de forma tal que el uso esperado es de la forma **./client <manag_addr> <manag_port>**. Una vez iniciado, consta de una interfaz de línea de comandos simple con 15 instrucciones habilitadas, todos case sensitive y enunciadas a continuación:

COMANDO	ARGUMENTO	DESCRIPCIÓN
help	-	Devuelve la lista de comandos disponibles.
historic	-	Devuelve la cantidad de conexiones históricas.
current	-	Devuelve la cantidad de conexiones actuales.
bytes	-	Devuelve la cantidad de bytes transferidos.
getbuff	-	Devuelve el tamaño del buffer utilizado.
setbuff	buffsize (unsigned int > 0)	Cambia el tamaño del buffer utilizado
gettimeout	-	Devuelve el timeout utilizado
settimeout	new_timeout (unsigned int > 0)	Cambia el timeout utilizado

geterror	-	Devuelve el file hacia donde se redirige el error
seterror	error_file (string)	Cambia el file hacia donde se redirige el error
getfilter	-	Devuelve el filtro que usa el transform
setfilter	filter_applied (string SIN comillas)	Cambia el filtro que usa el transform
filter?	-	Advierte si el filtro esta encendido o no
enablefilter	-	Enciende el filtro
disablefilter	-	Apaga el filtro

* Los bytes transferidos se calculan como los bytes que le llegan al origen + los bytes que le llegan a los clientes. No se cuenta lo que pasa por el filtro. Esto quiere decir que, si el cliente envía 10 bytes y el server responde con 50, que son transformados a 100, los bytes transferidos serían 110.

El cliente se encarga automáticamente de regular el req_id, la versión y el auth_id que se envían. El auth_id enviado se puede modificar cambiando la variable de entorno **SAP_AUTH**. Esta es la variable de entorno que revisa el proxy **en el momento de su ejecución inicial** para determinar el auth_id que aceptará.

En la mayoría de los casos, la respuesta a un request va a ser positiva y el usuario recibirá un mensaje de confirmación así como la data requerida de ser necesario. Sin embargo, hay un par de casos donde el request puede fallar y se lo informará adecuadamente al cliente:

- El auth_id configurado no coincide con el del server
- El server respondió con un req_id que no coincide con el request efectuado.
- El server no está respondiendo a los requests.
- Se quiso encender el filter sin que haya uno configurado
- Hubo un error en los parámetros

11. Ejemplos de configuración

11.1. Seteo de filtros

```
kali@kali:~/Documentos/Facultad/Protos/TP/Pop3Filter/manager_client$ ./client 127.0.0.1 9090
sap_client >> current
La cantidad de conexiones actuales es: 2
sap_client >> disablefilter
Filtro apagado
sap_client >> enablefilter
Filtro encendido
sap_client >> getfilter
El filtro utilizado es: ../bytes_tuffed.sh
sap_client >>

+OK Logged in.
retr 4
+OK 540 octets
Return-Path: <prueba@probando.com>
X-Original-To: kali@kali.kali
Delivered-To: kali@kali.kali
Received: from kali.kali (localhost [127.0.0.1])
        by kali.kali (Postfix) with ESMTP id 6527B165D2C
        for <kali@kali.kali>; Sun, 21 Nov 2021 11:44:16 -0300 (-03)
From: <prueba@probando.com>
To: <kali@kali.kali>
Subject: probando
MIME-version: 1.0
Content-type: text/plain; charset=UTF-8
Content-transfer-Encoding: quoted-printable
Message-Id: <20211121144427.6527B165D2C@kali.kali>
Date: Sun, 21 Nov 2021 11:44:16 -0300 (-03)
hola
.

RESP-CODES
PIPELINING
AUTH-RESP-CODE
list
+OK 5 messages:
1 537
2 545
3 541
4 540
5 694489434
.
retr 4
+OK 540 octets
X-Header: true
..hola
..
list
+OK 5 messages:
1 537
2 545
3 541
4 540
5 694489434
.
```

A la izquierda se ve el manejo del client. Se puede ver que hay dos conexiones y en ambas se hará un RETR 4 para obtener el cuarto mail. En orden cronológico:

1. En el directorio anterior al proyecto (en este caso ~/Documentos/Facultad/Protos/TP) se creó el archivo bytestuffed.sh (con permisos de ejecución chmod +x), con el siguiente contenido:


```
#!/bin/bash
cat > /dev/null
cat << EOF | unix2dos
X-Header: true

.hola
.
EOF
```
2. Se deshabilita el filtro para asegurarse que no hay ningún filtro activo, y en la consola 2 se hace un RETR 4. Se puede ver el mensaje sin filtrar.
3. Se habilita el filtro (../bytestuffed.sh, ../ al estar en el directorio anterior), y en la consola 3 se hace un RETR 4. Se puede ver el mensaje filtrado

11.2. Consulta de filter

```
sap_client >> disablefilter
Filtro apagado
sap_client >> filter?
El filtro esta apagado
sap_client >> enablefilter
Filtro encendido
sap_client >> filter?
El filtro esta encendido
sap_client >> █
```

Con el comando “filter?” puede consultarse si el filtro está encendido o no.

11.3. Uso de current y historic

```
kali@kali:~/Documentos/Facultad /Protos/TP/Pop3Filter/manager_client$ ./client 127.0.0.1 9090
sap_client >> historic
La cantidad de conexiones historicas es: 0
sap_client >> current
La cantidad de conexiones actuales es: 4
sap_client >> historic
La cantidad de conexiones historicas es: 4
sap_client >> current
La cantidad de conexiones actuales es: 0
sap_client >> current
La cantidad de conexiones actuales es: 4
sap_client >> historic
La cantidad de conexiones historicas es: 8
sap_client >> current
La cantidad de conexiones actuales es: 0
sap_client >> █

kali@kali:~$ nc -C foo 1110
+OK Dovecot (Debian) ready.
quit
+OK Logging out
kali@kali:~$ nc -C foo 1110
+OK Dovecot (Debian) ready.
quit
+OK Logging out
kali@kali:~$ █

kali@kali:~$ nc -C foo 1110
+OK Dovecot (Debian) ready.
quit
+OK Logging out
kali@kali:~$ nc -C foo 1110
+OK Dovecot (Debian) ready.
quit
+OK Logging out
kali@kali:~$ █

kali@kali:~$ nc -C foo 1110
+OK Dovecot (Debian) ready.
quit
+OK Logging out
kali@kali:~$ nc -C foo 1110
+OK Dovecot (Debian) ready.
quit
+OK Logging out
kali@kali:~$ █
```

1. Se comienza la prueba sin conexiones. Se muestra que el historic es 0.
2. Se abren 4 conexiones en simultáneo. Se ve que con el comando current se ven las 4 conexiones, y historic aumenta a 4.
3. Se cierran las 4 conexiones, llevando el current a 0.
4. Se abren 4 nuevas conexiones en simultáneo. Se ve que el current vuelve a ser 4 y historic aumenta a 8.
5. Se cierran las 4 conexiones y current vuelve a 0.

11.4. Uso de bytes

```
kali@kali:~/Documentos/Facultad/Protos/TP/Pop3Filter/manager_client$ ./client 127.0.0.1 9090
sap_client >> bytes
La cantidad de bytes transferidos es: 0
sap_client >> bytes
La cantidad de bytes transferidos es: 29
sap_client >> bytes
La cantidad de bytes transferidos es: 50
sap_client >> bytes
La cantidad de bytes transferidos es: 609
sap_client >> bytes
La cantidad de bytes transferidos es: 1168
sap_client >> []

+OK Logged in.
retr 4
+OK 540 octets
Return-Path: <prueba@probando.com>
X-Original-To: kali@kali.kali
Delivered-To: kali@kali.kali
Received: from kali.kali (localhost [127.0.0.1])
    by kali.kali (Postfix) with ESMTP id 6527B165D2C
    for <kali@kali.kali>; Sun, 21 Nov 2021 11:44:16 -03
00 (-03)
From: <prueba@probando.com>
To: <kali@kali.kali>
Subject: probando
MIME-version: 1.0
Content-type: text/plain; charset=UTF-8
Content-transfer-Encoding: quoted-printable
Message-Id: <20211121144427.6527B165D2C@kali.kali>
Date: Sun, 21 Nov 2021 11:44:16 -0300 (-03)

hola
.
retr 4
+OK 540 octets
Return-Path: <prueba@probando.com>
X-Original-To: kali@kali.kali
Delivered-To: kali@kali.kali
Received: from kali.kali (localhost [127.0.0.1])
    by kali.kali (Postfix) with ESMTP id 6527B165D2C
```

1. Comienza con la conexión apagada. Se ve que los bytes transferidos fueron 0.
2. Se inicia una conexión. Con los mensajes enviados al cliente para que éste sepa que la conexión fue correcta se envían en total 50 bytes.
3. El cliente realiza un RETR 4. Los bytes totales aumentan de 50 a 609 (es decir, se transfirieron 559 bytes entre origen y cliente).
4. El cliente vuelve a realizar un RETR 4. Los bytes deberían aumentar nuevamente 559 bytes, y se puede ver que los bytes totales aumentaron a 1168 bytes (es decir, 559 bytes).

11.5. Redirección de errores

```
kali@kali:~/Documentos/Facultad/Protos/TP/Pop3Filter/manager_client$ ./client 127.0.0.1 9090
sap_client >> geterror
La salida de error en filter es: /home/kali/Documentos/Facultad/Protos/errores
sap_client >> seterror /home/kali/Documentos/Facultad/Protos/error
La salida de error en filter fue actualizada
sap_client >> geterror
La salida de error en filter es: /home/kali/Documentos/Facultad/Protos/error
sap_client >> setfilter fallar
Filtro actualizado correctamente
sap_client >> enablefilter
Filtro encendido
sap_client >> []

kali@kali:~/Documentos/Facultad/Protos$ ls | grep error
errores
kali@kali:~/Documentos/Facultad/Protos$ ls | grep error
error
errores
kali@kali:~/Documentos/Facultad/Protos$ sudo cat error
sh: 1: fallar: not found

1 537
2 545
3 541
4 540
5 694489434
.
retr 4
```

1. Se inicia la conexión con el flag -e /home/kali/Documentos/Facultad/Protos/errores. De esta manera se indica que los errores irán al archivo “errores”.
2. Con el comando geterror se puede consultar el archivo de errores.
3. Con el comando seterror se cambia el destino de los errores. Se puede ver que hasta entonces no existe el archivo “error”.
4. Se impone un filtro inexistente “fallar” para forzar un error (el filtro debe ser un archivo .sh válido).

5. El cliente realiza un RETR 4. Al pedir por el mensaje se invoca al filtro, provocando un error.
6. Se crea el archivo “error”, indicando el error encontrado.

11.6. Cambio de timeout

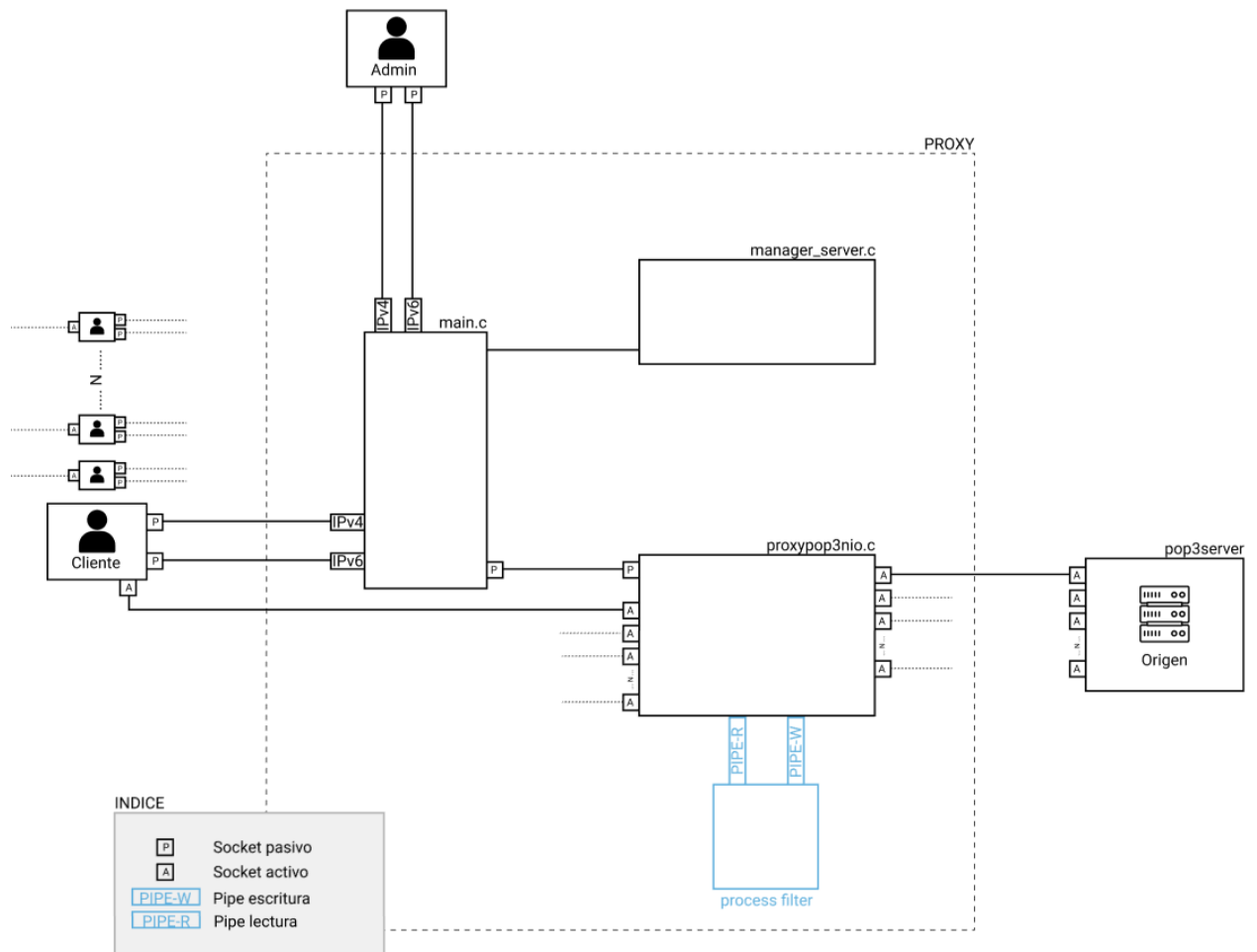
```
kali@kali:~/Documentos/Facultad/Protos/TP/Pop3Filter/manager_client$ ./client 127.0.0.1 9090
sap_client >> settimeout 10
Timeout actualizado correctamente
sap_client >> gettimeout
El timeout es: 10
```

```
kali@kali:~$ time nc -C 127.0.0.1 1110
+OK Dovecot (Debian) ready.

real    0m10.032s
user    0m0.003s
sys     0m0.000s
```

1. Con el comando settimeout se establece un timeout de 10 (se eligió un valor chico para que la prueba sea realizable).
2. Con el comando gettimeout se muestra que el timeout cambió con éxito.
3. Se inició la conexión con time para ver el tiempo de la conexión.
4. A los 10 segundos, la conexión finaliza automáticamente.

12. Diseño del proyecto



13. Casos de prueba

13.0. Convenciones

- ./pop3filter se refiere al binario con el proxy POP3.
- ./client se refiere al binario usado para obtener métricas de forma remota y obtener alguna configuración. Dicho binario se encuentra dentro del directorio manager_client del proyecto.
- Se limitó la memoria del entorno a la mínima para funcionar. Para esto se utilizó el siguiente comando:
`ulimit -v $((1024*1024*1024*32))`
- Para utilizar el servidor pop3.awk se debe dejar una copia del mismo (disponible en campus) en el directorio /tmp y ejecutar el siguiente comando:
`socat TCP4-LISTEN:9001,crlf,reuseaddr
SYSTEM:'/tmp/pop3.awk',pty,echo=0`
Para utilizarlo, ejecutar el proxy con el siguiente comando:
`$./pop3filter bar -p 9001`

13.1. Argumentos de las aplicaciones

Precondiciones:

- No se tiene corriendo ningún origen server en localhost.

13.1.0. Binding en puertos default correcto

Ejecutar

Terminal A: \$./pop3filter 127.0.0.1

Terminal B: \$ netstat -nlp | grep pop3filter

```
kali@kali:~/Documentos/Facultad/Protos/TP/Pop3Filter$ sudo netstat -nlp | grep pop3filter
tcp        0      0 0.0.0.0:1110          0.0.0.0:*            LISTEN      9183/./pop3filter
tcp6       0      0 :::1110              :::*                  LISTEN      9183/./pop3filter
udp        0      0 127.0.0.1:9090       0.0.0.0:*            9183/./pop3filter
udp6       0      0 :::1:9090            :::*                  9183/./pop3filter
```

Se verifica:

- Puerto TCP 1110 escuchando en todas las interfaces.
- Puerto UDP 9090 escuchando únicamente en localhost.
- Ejecución exitosa de pop3ctl (obtención de métricas o configuración) → se llama .client en manager_client

```
kali@kali:~/Documentos/Facultad/Protos/TP/Pop3Filter/manager_client$ ./client 127.0.0.1 9090
sap_client >> getbuff
El tamaño del buffer es: 2048
sap_client >> gettimeout
El timeout es: 200
sap_client >>
```

13.1.1. Cambio de puertos e interfaces default

Ejecutar

Terminal A: \$./pop3filter -p1111 -o 9091 -l ::1 -L0.0.0.0 127.0.0.1

Terminal B: \$ netstat -nlp | grep pop3filter

```
kali@kali:~/Documentos/Facultad/Protos/TP/Pop3Filter$ sudo netstat -nlp | grep pop3filter
[sudo] password for kali:
tcp6       0      0 :::1111              :::*                  LISTEN      7687/./pop3filter
udp        0      0 0.0.0.0:9091        0.0.0.0:*            7687/./pop3filter
```

Se verifica:

- Puerto TCP 1111 escuchando únicamente en localhost.
- Puerto UDP 9091 escuchando en todas las interfaces.
- Ejecución exitosa de pop3ctl (obtención de métricas o configuración).

```
kali@kali:~/Documentos/Facultad/Protos/TP/Pop3Filter/manager_client$ ./client 127.0.0.1 9091
sap_client >> getbuff
El tamaño del buffer es: 2048
sap_client >> gettimeout
El timeout es: 200
sap_client >>
```

13.2. Disponibilidad

13.2.0. Origin server (IPV4) no presta servicio

Precondiciones:

- En 127.0.0.1 puerto 110 no se encuentra ningún servidor tcp

```
kali@kali:~/Documentos/Facultad/Protos/TP/Pop3Filter/manager_client$ sudo netstat -nlp | grep 110
tcp        0      0 0.0.0.0:110        0.0.0.0:*        LISTEN      7842/./pop3filter
```

Ejecutar:

Terminal A: \$./pop3filter 127.0.0.1

Terminal B: \$ nc -C 127.0.0.1 1110

```
kali@kali:~/Documentos/Facultad/Protos/TP/Pop3Filter$ nc -C 127.0.0.1 1110
-ERR Connection refused.
```

Se verifica:

- En la terminal B debe aparecer un saludo negativo (-ERR Connection refused) y cerrar la conexión TCP.

13.2.1. Origin server (IPV6) no presta servicio

Precondiciones:

- No se tiene corriendo ningún origin server en localhost.

Ejecutar:

Terminal A: \$./pop3filter ::1

Terminal B: \$ nc -C 127.0.0.1 1110

```
kali@kali:~/Documentos/Facultad/Protos/TP/Pop3Filter$ nc -C 127.0.0.1 1110
-ERR Connection refused.
```

Se verifica:

- En la terminal B debe aparecer un saludo negativo (-ERR Connection refused) y cerrar la conexión TCP.

13.2.2. Origin server con múltiples direcciones IP (una falla)

Precondiciones:

- Origin server disponible en 127.0.0.1.
- El nombre tpe.proto.leak.com.ar resuelve a localhost y a una dirección IP que no está disponible

```
kali@kali:/etc$ cat hosts
127.0.0.1    localhost foo bar tpe.proto.leak.com.ar
127.0.1.1    kali.kali      kali
127.0.0.1    foo.pdc.lab foo.example.org
240.0.0.1    tpe.proto.leak.com.ar
```

Ejecutar:

Terminal A: \$./pop3filter tpe.proto.leak.com.ar

Terminal B: \$ nc -C 127.0.0.1 1110

```
kali@kali:~/Documentos/Facultad/Protos/TP/Pop3Filter$ ./pop3filter tpe.proto.leak.com.ar
INFO: main.c:307, Waiting for IPv4 proxy connections on socket 0
ERROR: main.c:293, Passive: bind failed
DEBUG: main.c:68, Unable to build proxy passive socket in IPv6
INFO: main.c:307, Waiting for IPv4 manager connections on socket 3
DEBUG: main.c:286, String address doesn't translate to IPv6
DEBUG: main.c:105, Unable to build manager passive socket in IPv6
INFO: ./proxy/proxypop3nio.c:286, Accepting connection from: 127.0.0.1:53166
DEBUG: ./proxy/proxypop3nio.c:303, Trying to resolve name: tpe.proto.leak.com.ar
DEBUG: ./proxy/proxypop3nio.c:416, origin socket = 5
DEBUG: ./proxy/proxypop3nio.c:427, Connecting in progress
INFO: ./proxy/proxypop3nio.c:493, Connection successful. Client Address: 127.0.0.1:53166; Origin Address: 127.0.0.1:110.
DEBUG: ./proxy/proxypop3nio.c:729, Hello read finished successfully
DEBUG: ./proxy/proxypop3nio.c:765, Hello finished successfully
```

```
kali@kali:~/Documentos/Facultad/Protos/TP/Pop3Filter$ nc -C 127.0.0.1 1110
+OK Dovecot (Debian) ready.
user kali
+OK
```

```
+OK Logged in.
list
+OK 2 messages:
1 537
2 545
kali@kali:~/Documentos/Facultad/Protos/TP/Pop3Filter$ nc -C 127.0.0.1 1110
+OK Dovecot (Debian) ready.
[]
```

Se verifica:

- Saludo +OK, y se puede interactuar con el origin server por medio del proxy (USER, PASS, LIST).
- Probando con más conexiones erróneas sigue funcionando correctamente:

```
kali@kali:/etc$ head hosts
127.0.0.1 localhost foo bar
127.0.1.1 kali.kali kali
127.0.0.1 foo.pdc.lab foo.example.org
240.0.0.1 tpe.proto.leak.com.ar
240.0.0.2 tpe.proto.leak.com.ar
240.0.0.3 tpe.proto.leak.com.ar
127.0.0.1 tpe.proto.leak.com.ar
```

```
kali@kali:~/Documentos/Facultad/Protos/TP/Pop3Filter
$ ./pop3filter tpe.proto.leak.com.ar
INFO: main.c:321, Waiting for IPv4 proxy connections
on socket 0
INFO: main.c:321, Waiting for IPV6 proxy connections
on socket 3
INFO: main.c:321, Waiting for IPv4 manager connections
on socket 4
DEBUG: main.c:300, String address doesn't translate
to IPv6
DEBUG: main.c:106, Unable to build manager passive s
ocket in IPv6
INFO: proxypop3nio.c:286, Accepting connection from:
127.0.0.1:52736
DEBUG: proxypop3nio.c:303, Trying to resolve name: t
pe.proto.leak.com.ar
DEBUG: proxypop3nio.c:416, origin socket = 6
DEBUG: proxypop3nio.c:427, Connecting in progress
INFO: proxypop3nio.c:493, Connection successful. Cli
ent Address: 127.0.0.1:52736; Origin Address: 127.0.
0.1:110.
DEBUG: proxypop3nio.c:731, Hello read finished succe
ssfully
DEBUG: proxypop3nio.c:767, Hello finished succesfull
y
█
```

```
kali@kali:~/Documentos/Facultad/Protos/TP/Pop3Filter$ nc
-C 127.0.0.1 1110
+OK Dovecot (Debian) ready.
█
```

13.2.3. Comportamiento origin server resuelve DNS IPV6

Precondiciones:

- ipv6.leak.com.ar. es FQDN que resuelve una dirección IPV6 donde está corriendo el origin server.

```
kali@kali:/etc$ tail -n 4 hosts
# The following lines are desirable for IPv6 capable hosts
::1      localhost ip6-localhost ip6-loopback ipv6.leak.com.ar
```

Ejecutar:

Terminal A: \$./pop3filter ipv6.leak.com.ar

Terminal B: \$ nc -C foo 1110

```
kali@kali:~/Documentos/Facultad/Protos/TP/Pop3Filter$ ./pop3filter ipv6.leak.com.ar
INFO: main.c:307, Waiting for IPv4 proxy connections on socket 0
ERROR: main.c:293, Passive: bind failed
DEBUG: main.c:68, Unable to build proxy passive socket in IPv6
INFO: main.c:307, Waiting for IPv4 manager connections on socket 3
DEBUG: main.c:286, String address doesn't translate to IPv6
DEBUG: main.c:105, Unable to build manager passive socket in IPv6
INFO: ./proxy/proxypop3nio.c:286, Accepting connection from: 127.0.0.1:53210
DEBUG: ./proxy/proxypop3nio.c:303, Trying to resolve name: ipv6.leak.com.ar
DEBUG: ./proxy/proxypop3nio.c:416, origin socket = 5
DEBUG: ./proxy/proxypop3nio.c:427, Connecting in progress
INFO: ./proxy/proxypop3nio.c:493, Connection successful. Client Address: 127.0.0.1:53210; Origin Address: ::1:110.
DEBUG: ./proxy/proxypop3nio.c:729, Hello read finished successfully
DEBUG: ./proxy/proxypop3nio.c:765, Hello finished successfully
```

```
kali@kali:~/Documentos/Facultad/Protos/TP/Pop3Filter$ nc -C foo 1110
+OK Dovecot (Debian) ready.
user kali
+OK
```

```
+OK Logged in.
list
+OK 2 messages:
1 537
2 545
.
Verificar adicionalmente con wireshark que los paquetes entre el proxy y el
servidor origen se transfieren vía IPv6
```

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	70	53210 → 1110 [PSH, ACK] Seq=1 Ack=1 Win=512 Len=4 TSval=2...
2	0.000024959	127.0.0.1	127.0.0.1	TCP	66	1110 → 53210 [ACK] Seq=1 Ack=5 Win=512 Len=0 TSval=264208...
3	0.000057035	127.0.0.1	127.0.0.1	TCP	68	53210 → 1110 [PSH, ACK] Seq=5 Ack=1 Win=512 Len=2 TSval=2...
4	0.000061973	127.0.0.1	127.0.0.1	TCP	66	1110 → 53210 [ACK] Seq=1 Ack=7 Win=512 Len=0 TSval=264208...
5	0.000198018	:::1	:::1	POP	92	C: list
6	0.000264962	:::1	:::1	TCP	86	110 → 35884 [ACK] Seq=1 Ack=7 Win=512 Len=0 TSval=4084685...
7	0.000376108	:::1	:::1	POP	120	S: +OK 2 messages:
8	0.000384632	:::1	:::1	TCP	86	35884 → 110 [ACK] Seq=7 Ack=35 Win=512 Len=0 TSval=408468...
9	0.000435122	127.0.0.1	127.0.0.1	TCP	100	1110 → 53210 [PSH, ACK] Seq=1 Ack=7 Win=512 Len=34 TSval=...
10	0.000439626	127.0.0.1	127.0.0.1	TCP	66	53210 → 1110 [ACK] Seq=7 Ack=35 Win=512 Len=0 TSval=26420...

Se verifica:

- La presencia de un saludo positivo y que se puede interactuar con el servidor POP3.
- Se puede verificar adicionalmente con wireshark que los paquetes entre el proxy y el servidor origen se transfieren vía IPv6.

13.2.4. Proxy Server como Origin Server (recursividad)

Precondiciones:

- Terminal A: \$./pop3filter -P 1110 localhost

```
kali@kali:~/Documentos/Facultad/Protos/TP/Pop3Filter$ ./pop3filter -P 1110 localhost
INFO: main.c:307, Waiting for IPv4 proxy connections on socket 0
ERROR: main.c:293, Passive: bind failed
DEBUG: main.c:68, Unable to build proxy passive socket in IPv6
INFO: main.c:307, Waiting for IPv4 manager connections on socket 3
DEBUG: main.c:286, String address doesn't translate to IPv6
DEBUG: main.c:105, Unable to build manager passive socket in IPv6
```

Pasos a seguir:

Terminal B: \$ nc -C foo 1110

```
INFO: ./proxy/proxypop3nio.c:286, Accepting connection from: 127.0.0.1:53240
DEBUG: ./proxy/proxypop3nio.c:303, Trying to resolve name: localhost
DEBUG: ./proxy/proxypop3nio.c:416, origin socket = 5
DEBUG: ./proxy/proxypop3nio.c:427, Connecting in progress
ERROR: ./proxy/proxypop3nio.c:480, Problem connecting to origin server in on_connection-ready
DEBUG: ./proxy/proxypop3nio.c:1387, Sending error to client: -ERR Connection refused.
```

```
kali@kali:~/Documentos/Facultad/Protos/TP/Pop3Filter$ nc -C foo 1110
-ERR Connection refused.
```

Se verifica:

- Una implementación consciente del problema detecta la recursión antes de conectarse y responde con un mensaje de -ERR.

13.2.5. Performance e inmutabilidad de bytes

Precondiciones:

- Origin server disponible en el host bar.
- Usuario con al menos un correo electrónico.
- Terminal A: ./pop3filter bar

Se trabajará con el [Anexo - Mensaje “Grande”](#) para tener mejores mediciones (es el mail 5).

En primer lugar se calcula primero la línea base yendo directo contra el servidor pop3:

```
$ time printf 'USER foo\nPASS foo\nRETR 5\n'|nc -C bar 110|pv|tail -n +5|head -n -1|sha256sum
```

```
ba848c96631aba2e1725633ace608fd0521d125250b730905c14215100fabecd -
real 10m12,534s
```

```
user 0m2,928s
```

```
sys 0m3,429s
```

```
ba848c96631aba2e1725633ace608fd0521d125250b730905c14215100fabecd -
real 10m12,534s
user 0m2,928s
sys 0m3,429s
```

```
$ time printf 'USER foo\nPASS foo\nRETR 5\n'|nc -C foo 1110|tail -n +5|head -n -1|sha256sum
```

```
5e95fc3477fba33b6f9574bad93bbe4c4a55593f049a3cde644546fceaad3916 -
```

```
real 7m0,495s
```

```
user 0m3,156s
```

```
sys 0m3,982s
```

```
5e95fc3477fba33b6f9574bad93bbe4c4a55593f049a3cde644546fceaad3916 -
user 0m3,156s
real 7m0,495s
user 0m3,156s
sys 0m3,982s
```

Se verifica:

- Al calcular el checksum con el proxy, coincide con el checksum del mensaje original calculado en [Anexo - Mensaje “Grande”](#).
- El tiempo de transferencia está dentro de los parámetros esperados.

13.2.6. Concurrencia

Precondiciones:

- Mismas que el caso [13.2.5](#).

Se realizan los mismos pasos que en el caso [13.2.5](#) pero en simultáneo en 4 terminales (se corrió en 6 consolas en simultáneo, se muestra la salida de 4).

```
5e95fc3477fba33b6f9574bad93bbe4c4a55593f049a3cde644546fceaad3916 -
real 9m48,822s
user 0m4,498s
sys 0m7,101s
```

```
5e95fc3477fba33b6f9574bad93bbe4c4a55593f049a3cde644546fceaad3916 -
real 9m49,427s
user 0m4,915s
sys 0m8,218s
```

```
5e95fc3477fba33b6f9574bad93bbe4c4a55593f049a3cde644546fceaad3916 -
real 9m48,229s
user 0m4,576s
sys 0m6,900s
```

```
5e95fc3477fba33b6f9574bad93bbe4c4a55593f049a3cde644546fceaad3916 -
real 12m47,340s
user 0m4,743s
sys 0m7,718s
```

```
5e95fc3477fba33b6f9574bad93bbe4c4a55593f049a3cde644546fceaad3916
```

```
5e95fc3477fba33b6f9574bad93bbe4c4a55593f049a3cde644546fceaad3916
```

5e95fc3477fba33b6f9574bad93bbe4c4a55593f049a3cde644546fceaad3916

5e95fc3477fba33b6f9574bad93bbe4c4a55593f049a3cde644546fceaad3916

Se verifica:

- El sha256 coincide en todas las ejecuciones, y a su vez coincide con el sha256 del archivo original.
- El tiempo de transferencia utilizando el proxy de manera concurrente está dentro del mismo orden que la transferencia utilizando el proxy con una sola conexión.

13.2.7. Desconexión repentina de clientes

Precondiciones:

- Origin server disponible en el host bar.
- Usuario con al menos un correo electrónico (lo llamaremos foo con contraseña foo).
- Terminal A: ./pop3filter bar

```
kali@kali:~/Documentos/Facultad/Protos/TP/Pop3Filter$ ./pop3filter bar
INFO: main.c:307, Waiting for IPv4 proxy connections on socket 0
ERROR: main.c:293, Passive: bind failed
DEBUG: main.c:68, Unable to build proxy passive socket in IPv6
INFO: main.c:307, Waiting for IPv4 manager connections on socket 3
DEBUG: main.c:286, String address doesn't translate to IPv6
DEBUG: main.c:105, Unable to build manager passive socket in IPv6
INFO: ./proxy/proxypop3nio.c:286, Accepting connection from: 127.0.0.1:53430
DEBUG: ./proxy/proxypop3nio.c:303, Trying to resolve name: bar
DEBUG: ./proxy/proxypop3nio.c:416, origin socket = 5
```

En Terminal B:

\$ nc -C foo 1110

+OK listo para probar

```
kali@kali:~/Documentos/Facultad/Protos/TP/Pop3Filter$ nc -C foo 1110
+OK Dovecot (Debian) ready.
[]
```

En Terminal C

kill -9 <pid de nc de Terminal B>

```
kali@kali:~/Documentos/Facultad/Protos/TP/Pop3Filter$ ps ax | grep nc
912 ?        Ssl      0:00 /usr/libexec/at-spi-bus-launcher
4900 pts/1    S+       0:00 nc -C foo 1110
4939 pts/2    S+       0:00 grep --color=auto nc
kali@kali:~/Documentos/Facultad/Protos/TP/Pop3Filter$ kill -9 4900
```

Se verifica:

- Netcat termina

- El proxy sigue ejecutando y recibiendo nuevas conexiones y no se está consumiendo todo el CPU.

```
kali@kali:~/Documentos/Facultad/Protos/TP/Pop3Filter$ nc -C foo 1110
+OK Dovecot (Debian) ready.
Terminado (killed)
kali@kali:~/Documentos/Facultad/Protos/TP/Pop3Filter$ nc -C foo 1110
+OK Dovecot (Debian) ready.
user kali
+OK
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
602	root	20	0	419516	129580	67576	S	9,3	2,6	1:43.07	Xorg
935	kali	20	0	401568	95192	66152	S	6,0	1,9	0:34.13	xfwm4
5000	kali	20	0	327768	41684	33508	S	4,3	0,8	0:00.32	xfce4-screensho
885	kali	20	0	154816	2764	2316	S	0,7	0,1	0:21.67	VBoxClient
110	root	-51	0	0	0	0	S	0,3	0,0	0:02.14	irq/18-vmwgfx
3718	kali	20	0	3019888	448552	165000	S	0,3	8,9	4:26.64	firefox-esr
4045	kali	20	0	3037316	361180	139776	S	0,3	7,1	1:53.94	Web Content
4755	root	20	0	0	0	0	I	0,3	0,0	0:01.41	kworker/0:1-events
4910	kali	20	0	2507460	191736	117280	S	0,3	3,8	0:06.78	Web Content

13.2.8. Desconexión repentina del origen server esperando comando

Precondiciones:

- Origin server disponible en el host bar.
- Usuario con al menos un correo electrónico (lo llamaremos foo con contraseña foo).

Terminal A:

./pop3filter bar

```
kali@kali:~/Documentos/Facultad/Protos/TP/Pop3Filter$ ./pop3filter bar
INFO: main.c:307, Waiting for IPv4 proxy connections on socket 0
ERROR: main.c:293, Passive: bind failed
DEBUG: main.c:68, Unable to build proxy passive socket in IPv6
INFO: main.c:307, Waiting for IPv4 manager connections on socket 3
DEBUG: main.c:286, String address doesn't translate to IPv6
DEBUG: main.c:105, Unable to build manager passive socket in IPv6
```

En Terminal B:

\$ nc -C foo 1110

+OK listo para probar

```
kali@kali:~/Documentos/Facultad/Protos/TP/Pop3Filter$ nc -C foo 1110
+OK Dovecot (Debian) ready.
```

En Terminal C

terminar el servidor pop

```
kali@kali:~/Documentos/Facultad/Protos/TP/Pop3Filter$ sudo systemctl
stop dovecot
kali@kali:~/Documentos/Facultad/Protos/TP/Pop3Filter$ sudo netstat -n
ltp | grep 110
tcp        0      0 0.0.0.0:1110          0.0.0.0:*            LISTEN
5196/./pop3filter
```

Se verifica:

- El servidor POP3 no está disponible
- El proxy sigue ejecutando y recibiendo nuevas conexiones y no se está consumiendo la mayoría del CPU.

```
ERROR: ./proxy/proxypop3nio.c:926, Read
ed 0 or error on client. Error: Interru
pted system call
ERROR: ./proxy/proxypop3nio.c:926, Read
ed 0 or error on client. Error: Transpo
rt endpoint is not connected
INFO: ./proxy/proxypop3nio.c:286, Accep
ting connection from: 127.0.0.1:53610
DEBUG: ./proxy/proxypop3nio.c:303, Tryi
ng to resolve name: bar
DEBUG: ./proxy/proxypop3nio.c:416, orig
in socket = 5
DEBUG: ./proxy/proxypop3nio.c:427, Conn
ecting in progress
ERROR: ./proxy/proxypop3nio.c:480, Prob
lem connecting to origin server in on_c
onnection-ready
DEBUG: ./proxy/proxypop3nio.c:1387, Sen
ding error to client: -ERR Connection r
efused.
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
602	root	20	0	437424	151352	72000	S	2,3	3,0	2:20.47	Xorg
935	kali	20	0	401568	95660	66388	S	2,0	1,9	0:53.87	xfwm4
5299	kali	20	0	327764	41792	33580	S	1,0	0,8	0:00.36	xfce4-screensho
5263	kali	20	0	9064	3572	3140	R	0,7	0,1	0:00.06	top
110	root	-51	0	0	0	0	S	0,3	0,0	0:02.57	irq/18-vmwgfx
885	kali	20	0	154816	2764	2316	S	0,3	0,1	0:32.70	VBoxClient

- La conexión de la Terminal B termina al enviar el siguiente comando.

```
kali@kali:~/Documentos/Facultad/Protos/TP/Pop3Filter$ nc -C foo 1110
+OK Dovecot (Debian) ready.
user kali
-ERR Shutting down
```

13.2.9. Desconexión repentina del origin server durante un RETR

Precondiciones:

- Origin server disponible en el host bar.
- Usuario con al menos un correo electrónico (lo llamaremos foo con contraseña foo).
- Terminal A: ./pop3filter bar

- Se obtendrá el mensaje más grande para tener tiempo de matar el servidor pop origen: [Anexo - Mensaje “Grande”](#).

En Terminal B:

```
$ nc -C foo 1110
```

En Terminal C:

```
systemctl stop dovecot
```

```
kali@kali:~/Documentos/Facultad/Protos/TP/Pop3Filter$ ./pop3filter bar
INFO: main.c:328, Waiting for IPv4 proxy connections on socket 0
INFO: main.c:328, Waiting for IPV6 proxy connections on socket 3
INFO: main.c:328, Waiting for IPv4 manager connections on socket 4
INFO: main.c:328, Waiting for IPV6 manager connections on socket 5
INFO: proxypop3nio.c:286, Accepting connection from: 127.0.0.1:54718
DEBUG: proxypop3nio.c:303, Trying to resolve name: bar
DEBUG: proxypop3nio.c:418, origin socket = 7
DEBUG: proxypop3nio.c:429, Connecting in progress
INFO: proxypop3nio.c:495, Connection successful. Client Address: 127.0.0.1:54718; Origin Address: 127.0.0.1:1110
DEBUG: proxypop3nio.c:730, Hello read finished successfully
DEBUG: proxypop3nio.c:766, Hello finished successfully
DEBUG: proxypop3nio.c:1010, Logged user: kali
[]
```

```
kjXQ9Rvc2jXaIUQD+HBKaK
hL73BHHRcLCuTvK6qAMsro91joomsty0yfw0mcSMIT42L1sNfZt1Qt
DuIGFVZm1gcWCgNgJckok0
T7weoURKC68aAimutxMpKP0b2Luq4YPaPW0BSAfnLB1SNcxWS9rBZh
6B32sxXHePmjjXRdvsqvx
Qu6nAZV2uPhgw4IpikefoJEg378xiJkgHrFIrA+u8XhkdPY9m8SG
gS+KRNxdoKwj+nbqPdBX8E
/X7Rep50Sww0UozGKEhELn8cP8fR9IfPfnetkiEy8Bo/fSA7yodz0
DkuXDP+L/g058b2sUsewT3
wQOIawPHJEgNI1XJXHJHDXEdm4a7lgb9kuyKtTDDutkrB6w//wveYt
I8gw4pVKJa3+01F1s+0Kl
qYxRkywIdE4wP0aTV3/6oi2XUsDecs/Xb+r4p1avvdbvEuTQnk3PT3
ee0BD3NhnYkali@kali:~/Documentos/Facultad/Protos/TP/Pop3Filter$
```

```
kali@kali:~$ sudo systemctl stop dovecot
[sudo] password for kali:
kali@kali:~$ netstat -ntlp | grep dovecot
(Not all processes could be identified, non-owned processes info
will not be shown, you would have to be root to see it all.)
kali@kali:~$ sudo netstat -ntlp | grep dovecot
kali@kali:~$
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
566	root	20	0	480428	167028	72620	S	4,3	3,3	5:22.21	Xorg
944	kali	20	0	411012	67024	42880	S	2,7	1,3	2:28.17	xfwm4
10015	kali	20	0	327772	41984	33796	S	2,0	0,8	0:00.28	xfce4-screensho
896	kali	20	0	154816	2084	2084	S	1,0	0,0	1:30.93	VBoxClient
4454	kali	20	0	2987764	382552	81068	S	0,7	7,6	4:07.58	Web Content

Se verifica:

- El servidor POP3 no está disponible.
- El proxy sigue ejecutando y recibiendo nuevas conexiones y no se está consumiendo la mayoría del CPU.
- Se termina la conexión de la Terminal B.

13.2.10. Lecturas parciales desde el cliente

Pre-condiciones:

- Todas las transformaciones y opciones apagadas
- Origin server pop3

Pasos a seguir.

Se trabajará en POP3 enviando comandos cada tecla con una separación de medio segundo, logrando que cada byte se envíe en su propio paquete IP.

```
$ stty -icanon && nc -C 127.0.0.1 1110
```

```
kali@kali:~/Documentos/Facultad/Protos/TP/Pop3Filter$ stty -icanon &&
nc -C 127.0.0.1 1110
+OK Dovecot (Debian) ready.
user kali
+OK
```

```
retr 1
+OK 537 octets   12   13   14   15   16   17   18
Return-Path: <lcatolino@itba.edu.ar>
X-Original-To: kali@kali.kali
Delivered-To: kali@kali.kali
Received: from kali.kali (localhost [127.0.0.1])
  by kali.kali (Postfix) with ESMTP id 99EE1161639
  for <kali@kali.kali>; Wed, 10 Nov 2021 16:12:02 -0300 (-03)
From: <kali@kali.kali>
To: <kali@kali.kali>
Message-Id: <20211110191248.99EE1161639@kali.kali>
Date: Wed, 10 Nov 2021 16:12:02 -0300 (-03)

DATA
MIME-version: 1.0
Content-type: text/plain; charset= UTF-8
Content-transfer-Encoding: quoted-printable

Mail de prueba
.
quit
+OK Logging out.
```

Se verifica:

- Se observa la respuesta de forma correcta

13.3. Uso del proxy POP3

13.3.1. Comportamiento origin server que da servicio comandos básicos

Precondiciones:

- Origin server disponible en el host bar.
- Usuario con al menos un correo electrónico
- Terminal A: ./pop3filter bar


```
kali@kali:~/Documentos/Facultad/Protos/TP/Pop3Filter$ ./pop3filter bar
INFO: main.c:321, Waiting for IPv4 proxy connections on socket 0
INFO: main.c:321, Waiting for IPv6 proxy connections on socket 3
INFO: main.c:321, Waiting for IPv4 manager connections on socket 4
DEBUG: main.c:300, String address doesn't translate to IPv6
DEBUG: main.c:106, Unable to build manager passive socket in IPv6
```

Terminal B: nc -C foo 1110

```
kali@kali:~/Documentos/Facultad/Protos/TP/Pop3Filter$ nc -C foo 1110
+OK Dovecot (Debian) ready.
CAPA
+OK
CAPA
TOP
UIDL
RESP-CODES
PIPELINING
AUTH-RESP-CODE
STLS
USER
SASL PLAIN
.
CAPA 123
+OK
CAPA
TOP
UIDL
RESP-CODES
PIPELINING
AUTH-RESP-CODE
STLS
USER
SASL PLAIN
.
USER kali
+OK
```

Chat

① Los chats del documento no se guardarán

 **LUCAS CATOLINO**

Buenas, estás haciendo los tests?
Pudiste generar el mensaje "grande"


```

+OK Logged in.
NOOP
+OK
STAT
+OK 2 1082
LIST
+OK 2 messages:
1 537
2 545
.
+OK 2 messages:
LIST 1
+OK 1 537
LIST 5000
-ERR There's no message 5000.
DELE 1
+OK Marked to be deleted.
RSET 0
+OK
RETR 1
+OK 537 octets
Return-Path: <lcatolino@itba.edu.ar>
X-Original-To: kali@kali.kali
Delivered-To: kali@kali.kali
Received: from kali.kali (localhost [127.0.0.1])
        by kali.kali (Postfix) with ESMTP id 99EE1161639
        for <kali@kali.kali>; Wed, 10 Nov 2021 16:12:02 -0300 (-03)
FRom: <kali@kali.kali>

```

```

To: <kali@kali.kali>
Message-Id: <20211110191248.99EE1161639@kali.kali>
Date: Wed, 10 Nov 2021 16:12:02 -0300 (-03)

DATA
MIME-version: 1.0
Content-type: text/plain; charset= UTF-8
Content-transfer-Encoding: quoted-printable

Mail de prueba
.
RSET
+OK

```

```

TOP 1 1
+OK
Return-Path: <lcatolino@itba.edu.ar>
X-Original-To: kali@kali.kali
Delivered-To: kali@kali.kali
Received: from kali.kali (localhost [127.0.0.1])
        by kali.kali (Postfix) with ESMTP id 99EE1161639
        for <kali@kali.kali>; Wed, 10 Nov 2021 16:12:02 -0300 (-03)
FRom: <kali@kali.kali>
To: <kali@kali.kali>
Message-Id: <20211110191248.99EE1161639@kali.kali>
Date: Wed, 10 Nov 2021 16:12:02 -0300 (-03)

```

```

DATA
.

```

```

UIDL
+OK 02 -0300 (-03)
1 000000026160a301
2 000000036160a301
.
UIDL 1
+OK 1 000000026160a301
QUIT
+OK Logging out.

```

Se verifica:

- Todos los comandos reciben la respuesta esperada

13.3.2. Pipelining cliente

Precondiciones:

- Mismas que el caso [13.3.1.](#)

En Terminal B:

```
$ printf 'USER foo\nPASS foo\nLIST\nTOP 1 1\nQUIT\n' | nc -C foo 1110
```

```

+OK Dovecot (Debian) ready.
+OK
+OK Logged in.
+OK 2 messages:
1 537
2 545
.
+OK
Return-Path: <lcatolino@itba.edu.ar>
X-Original-To: kali@kali.kali
Delivered-To: kali@kali.kali
Received: from kali.kali (localhost [127.0.0.1])
        by kali.kali (Postfix) with ESMTP id 99EE1161639
        for <kali@kali.kali>; Wed, 10 Nov 2021 16:12:02 -0300 (-03)
From: <kali@kali.kali>
To: <kali@kali.kali>
Message-Id: <20211110191248.99EE1161639@kali.kali>
Date: Wed, 10 Nov 2021 16:12:02 -0300 (-03)

DATA
.
+OK Logging out.

```

Se verifica:

- Todos los comandos reciben la respuesta esperada

13.3.3. Trailing spaces no afectan a los comandos

Similar a [13.3.2. Pipelining Cliente](#) pero agregando espacios al final de un comando.

Se espera la misma salida que dicho caso.

```
$ printf 'USER foo\nPASS foo\nLIST \nTOP 1 1\nQUIT \n' | nc -C foo 1110
```

```
+OK Dovecot (Debian) ready.  
+OK  
+OK Logged in.  
+OK 2 messages:  
1 537  
2 545  
bandos  
-ERR Noise after size:  
+OK Logging out.
```

Se verifica:

- El funcionamiento de los comandos es el mismo que en dovecot. Los comandos con espacios al final no afectan al proxy.

13.4. Pipelining

13.4.1. Pipelining con origin server que no soporta pipelining (de forma declarada)

Precondiciones:

- Servidor origen ejecutando pop3.awk
socat TCP4-LISTEN:9001,crlf,reuseaddr SYSTEM:'/tmp/pop3.awk',pty,echo=0
- Terminal A: \$./pop3filter bar -p 9001

Ejecutar:

```
printf "CAPA\nUSER foo\nPASS foo\nLIST\n"| nc -C foo 9001
```

```
+OK hola!  
+OK Mis capacidades son:  
CAPA  
USER  
SINPIPELINING  
.  
+OK Hola kali  
+OK Gracias no se la cuento a nadie  
+OK Tengo un mensaje  
1 20  
.
```

Se verifica:

- La salida es la esperada

13.4.2. Pipelining hacia origin server (declara que lo soporta)

Precondiciones:

- Servidor origen ejecutando soporta CAPA y PIPELINING
- Terminal A: \$./pop3filter bar
- Terminal B: sudo tcpdump -s0 -A1 'tcp port 110'

Ejecutar:

```
printf "USER foo\nPASS foo\nLIST\n"| nc -C foo 1110
```

```
+OK Dovecot (Debian) ready.  
+OK  
+OK Logged in.  
+OK 5 messages:  
1 537  
2 545  
3 541  
4 540  
5 694489434  
.  
Comunicación  
█
```

Se verifica:

- Funciona

13.5. Integración de la Transformaciones de mensajes

13.5.1. Transformación utilizando cat

Precondiciones:

- Origin server disponible en el host bar.
- Usuario con al menos un correo electrónico
- Terminal A: ./pop3filter -t 'touch /tmp/1 && cat' bar

La intención del caso es corroborar que se ejecuta el proceso externo para transformar. Utilizamos cat que no realiza ninguna modificación sobre el mensaje.

En la terminal B asegurarse de que no exista el archivo /tmp/1 (rm -rf /tmp/1), y se obtienen los mensajes.

```
kali@kali:~/Documentos/Facultad/Protos/TP/Pop3Filter$ rm
-rf /tmp/1
kali@kali:~/Documentos/Facultad/Protos/TP/Pop3Filter$ nc
-C 127.0.0.1 1110
+OK Dovecot (Debian) ready.
user kali
+OK
+OK Logged in.
retr 1
+OK 537 octets
Return-Path: <lcatolino@itba.edu.ar>
X-Original-To: kali@kali.kali
Delivered-To: kali@kali.kali
Received: from kali.kali (localhost [127.0.0.1])
    by kali.kali (Postfix) with ESMTP id 99EE1161639
    for <kali@kali.kali>; Wed, 10 Nov 2021 16:12:02 -
0300 (-03)
From: <kali@kali.kali>
To: <kali@kali.kali>
Message-Id: <20211110191248.99EE1161639@kali.kali>
Date: Wed, 10 Nov 2021 16:12:02 -0300 (-03)

DATA
MIME-version: 1.0
Content-type: text/plain; charset= UTF-8
Content-transfer-Encoding: quoted-printable

Mail de prueba
.

kali@kali:/tmp$ ls
1
```

Se verifica:

- Se crea el archivo /tmp/1.
- No existan diferencias en el mensaje

13.5.2. Seteo de variables de entorno

Precondiciones:

- Origin server disponible en el host bar.
- Usuario con al menos un correo electrónico (lo llamaremos foo con contraseña foo). Allí están los mensajes de prueba.
- Terminal A: `./pop3filter -t 'bash ../envs.sh' bar`

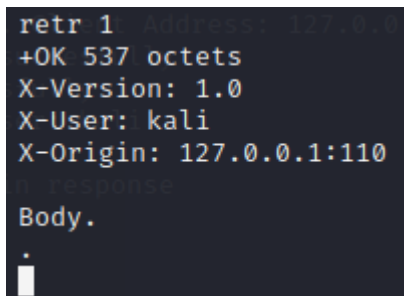
Donde envs.sh:

```
#!/bin/bash
cat << EOF | unix2dos
X-Version: $POP3FILTER_VERSION
X-User: $POP3_USERNAME
X-Origin: $POP3_SERVER
```

Body.

EOF

cat > /dev/null



```
retr 1
+OK 537 octets
X-Version: 1.0
X-User: kali
X-Origin: 127.0.0.1:110

Body.
.
```

Se verifica:

- Las variables de entorno son correctas.

13.5.3. Respuesta lazy

Precondiciones:

- Origin server disponible en el host bar.
- Usuario con al menos un correo electrónico.
- Terminal A: `./pop3filter -t 'bash lazy.sh' bar`

Donde lazy.sh:

```
#!/bin/bash
cat > /tmp/foo
cat /tmp/foo
```

Se verifica:

- El proxy no espera un flujo de datos específico: el proceso transformador consume todos los bytes para recién luego emitirlos (no se puede visualizar en imágenes, pero al hacer el retr del mensaje grande no se muestra nada por un rato hasta que se carga el archivo foo, y recién ahí lo empieza a mostrar).
- En el cliente el mensaje no contiene diferencias (con un mensaje chico se ve a ojo).
- En el mensaje grande, los checksum coinciden:

```
kali@kali:/tmp$ sha256sum foo
5e95fc3477fba33b6f9574bad93bbe4c4a55593f049a3cde644546fceaad3916  foo

kali@kali:~/Maildir/cur$ sha256sum i_big.mbox\:2\S
5e95fc3477fba33b6f9574bad93bbe4c4a55593f049a3cde644546fceaad3916  i_big.mbox:2,S

kali@kali:/tmp$ sha256sum foo | grep 5e95fc3477fba33b6f9574bad93bbe4c4a55593f049a3cde644546fceaad3916
5e95fc3477fba33b6f9574bad93bbe4c4a55593f049a3cde644546fceaad3916  foo
```

13.5.4. Byte Stuffed

Precondiciones:

- Origin server disponible en el host bar
- Usuario con al menos un correo electrónico
- Terminal A: `./pop3filter -t ../bytestuffed.sh bar`

Donde bytestuffed.sh:

```
#!/bin/bash
cat > /dev/null
cat << EOF | unix2dos
X-Header: true

.hola
.
EOF
```

Nota: el archivo bytestuffed.sh se encuentra en el directorio anterior al pop3filter.

```
kali@kali:~/Documentos/Facultad/Protos/TP/Pop3Filter
$ ./pop3filter -t ../bytestuffed.sh bar
INFO: main.c:321, Waiting for IPv4 proxy connections
on socket 0
INFO: main.c:321, Waiting for IPV6 proxy connections
on socket 3
INFO: main.c:321, Waiting for IPv4 manager connections
on socket 4
DEBUG: main.c:300, String address doesn't translate
to IPv6
DEBUG: main.c:106, Unable to build manager passive s
ocket in IPv6
INFO: proxypop3nio.c:286, Accepting connection from:
127.0.0.1:53988
DEBUG: proxypop3nio.c:303, Trying to resolve name: b
ar
DEBUG: proxypop3nio.c:416, origin socket = 6
DEBUG: proxypop3nio.c:427, Connecting in progress
INFO: proxypop3nio.c:493, Connection successful. Cli
ent Address: 127.0.0.1:53988; Origin Address: 127.0.
0.1:110.
DEBUG: proxypop3nio.c:731, Hello read finished succe
ssfully
DEBUG: proxypop3nio.c:767, Hello finished succesfull
y
DEBUG: proxypop3nio.c:1012, Logged user: kali
DEBUG: proxypop3nio.c:978, Filter is interest in res
ponse
DEBUG: proxypop3nio.c:792, Starting filter
DEBUG: proxypop3nio.c:1060, Closing filter
```

- Con filter:

```
+OK Logged in.
retr 4
+OK 540 octets
X-Header: true

.. hola
..
.
█
```

- Sin filter:


```
+OK Logged in.
retr 4
+OK 540 octets
Return-Path: <prueba@probando.com>
X-Original-To: kali@kali.kali
Delivered-To: kali@kali.kali
Received: from kali.kali (localhost [127.0.0.1])
    by kali.kali (Postfix) with ESMTP id 6527B165D2C
    for <kali@kali.kali>; Sun, 21 Nov 2021 11:44:16 -
0300 (-03)
From: <prueba@probando.com>
To: <kali@kali.kali>
Subject: probando
MIME-version: 1.0
Content-type: text/plain; charset=UTF-8
Content-transfer-Encoding: quoted-printable
Message-Id: <20211121144427.6527B165D2C@kali.kali>
Date: Sun, 21 Nov 2021 11:44:16 -0300 (-03)

hola
.
```

Se verifica:

- El filtro funciona de la forma esperada.

Anexo - Mensaje “Grande”

0. Descargar el archivo mensajes.tar.gz disponible en campus, descomprimirlo en un directorio y posicionar la consola en él. Para este ejemplo se trabajó en el directorio ~/Documentos/Facultad/Protos/Mensaje.

1. Descargar archivo de prueba.

```
$wget
```

```
https://archives.fedoraproject.org/pub/archive/fedora/linux/releases/26/Server/x86_64/iso/Fedora-Server-netinst-x86_64-26-1.5.iso
```

2. Verificar descarga exitosa. La salida del siguiente comando debe coincidir.

```
kali@kali:~/Documentos/Facultad/Protos/Mensaje$ sha256sum Fedora-Server-netinst-x86_64-26-1.5.iso | grep e260921ef5c7bd5ee2a7b2f2f1156af6483014c73984e4cf37f2b6690e0155e5
e260921ef5c7bd5ee2a7b2f2f1156af6483014c73984e4cf37f2b6690e0155e5  Fedora-Server-netinst-x86_64-26-1.5.iso
```

Se ve que coinciden. La descarga fue exitosa.

3. Verificar que el comando base64 genere líneas de 76 caracteres. De no ser así modificar los comandos que aparecen en la guía para que se genere la misma salida y de esta forma sea sencilla la comparación. Típicamente la opción -w controla el ancho de cada línea.

Una línea es:

```
ZOMwyIeOmaqjWb18iy6nDGI8rIeKzuHvyyjfPQU0H22SmTc1ed4jobutsQ7nECvGo4qW2FFn9tc → Son 76 caracteres
```

4. Armado del mensaje.

```
$ cat i_big.head > i_big.mbox
```

```
$ base64 < Fedora-Server-netinst-x86_64-26-1.5.iso | unix2dos >> i_big.mbox
```

```
$ cat i_big.tail >> i_big.mbox
```

```
$ sha256sum i_big.mbox
```

```
5e95fc3477fba33b6f9574bad93bbe4c4a55593f049a3cde644546fceaad3916
```

```
kali@kali:~/Documentos/Facultad/Protos/Mensaje$ sha256sum i_big.mbox | grep 5e95fc3477fba33b6f9574bad93bbe4c4a55593f049a3cde644546fceaad3916
5e95fc3477fba33b6f9574bad93bbe4c4a55593f049a3cde644546fceaad3916 i_big.mbox
```

```
$ ls -l i_big.mbox
```

```
$ wc -l i_big.mbox
```

```
8903731 i_big.mbox → respuesta esperada correcta
```

Checksum del mail:

```
$ printf 'USER foo\nPASS foo\nRETR 1\n'|nc -C bar 110|tail -n +5|head -n -1 |sha256sum
ba848c96631aba2e1725633ace608fd0521d125250b730905c14215100fabecd -
```