
# Trabajo Práctico Especial

## Materia: 72.07 - Protocolos de Comunicación

Profesores:

- Codagnone, Juan Francisco
- Garberoglio, Marcelo Fabio
- Kulesz, Sebastian

Autores:

- Cupitó, Felipe - fcupito@itba.edu.ar - 60058
- De Luca, Juan Manuel - jdeluca@itba.edu.ar - 60103
- Kim, Azul Candelaria - azkim@itba.edu.ar - 60264
- Maggioni Duffy, Faustino - fmaggioniduffy@itba.edu.ar - 64578

# Índice

# 1. Introducción

Según lo definido en el RFC 1928, el protocolo SOCKS Versión 5, es una extensión de la versión 4, aumentando su seguridad, incluyendo la posibilidad de permitir conexiones UDP, además de extender el framework para habilitar esquemas de autenticación más robustos. A lo largo de la materia de Protocolos de Comunicación, el protocolo SOCKS Versión 5 es uno de los temas que se tratan, y para este trabajo práctico se pide la implementación del mismo en una aplicación, además del diseño, creación, e implementación de un protocolo ideado por el grupo para monitorear un servidor SOCKS5. En este documento se detalla el proceso mediante el cual se llevó a cabo dicho desarrollo, así como también algunos de los problemas encontrados, las limitaciones de la aplicación, sus posibles extensiones, entre otras cosas.

## 2. Aplicación desarrollada

La aplicación desarrollada consiste en un servidor Proxy Socks que sigue la versión 5 de dicho protocolo, descrita en el RFC 1928, de manera no bloqueante.

### 2.a. Autenticación

Inicialmente se presentan las opciones de autenticación posibles, en un primer paquete llamado “Hello”. Es aquí donde comienza la negociación, el cliente presenta una lista con métodos de autenticación y el servidor puede o no ponerse de acuerdo, es decir seleccionar uno de los métodos provistos, y en base a eso aceptar o rechazar la conexión enviando el mensaje de response correspondiente. Existen otros métodos de autenticación descritos en SOCKS5 pero en nuestra implementación los dos permitidos son los especificados a continuación. Puede seleccionarse el método de no autenticación, en el cual no será necesario especificar usuario y contraseña. Si el método de autenticación es USER/PASS el usuario debe iniciar una sub negociación específica y autenticarse en el servidor de acuerdo a lo descrito en el RFC 1929.

### 2.b. Manejo de conexiones

Luego de establecer la sesión en el servidor el usuario manda una request. Hay varios tipos de request pero en nuestro caso la única válida es ‘Connect’, que permite conectarse a una IP a través de nuestro servidor proxy. Para ello se debe especificar la dirección IP destino, junto con su tipo, además del puerto destino. Nuestro servidor permite conexiones y redirecciones a direcciones IPv4, IPv6 y nombres de dominio.

Todos los requests de conexión se informarán mediante logs en la salida estándar. Para ver el formato de los logs, consultar la imagen 11 en el Anexo.

Una vez establecida la conexión con el servidor de origen, cada vez que el cliente envía un paquete, llegan al servidor mediante el proxy. También aplica de manera viceversa.

Para implementar esto, se utilizó un selector (código provisto por la cátedra) en el cual tienen el registro de los file descriptors de cada socket que se encuentre abierto. Hay que tener en cuenta que el límite de este es de 1024 file descriptors lo cual limitó la cantidad de conexiones posibles. El selector se encarga de iterar por cada uno y sólo cuando lo requieran, realizan las operaciones indicadas.

## 2.c. Métricas

Se requirió implementar mecanismos para la recolección de métricas que permitan monitorear el uso del servidor. Estas son la cantidad de conexiones históricas y concurrentes y la cantidad de bytes recibidos, enviados y transferidos. Estas métricas pueden ser accedidas mediante el protocolo Wannapprove. Para más información sobre este protocolo, proceder a la sección 4.

## 2.d. Sniffing

Para esta implementación se realizaron una serie de disectores de contraseñas para el protocolo POP3. Nuestro servidor mantiene un estado “spoofing” que está prendido o apagado. Cuando está prendido, nuestro servidor lee los intercambios de texto entre el cliente y el servidor al que se conecta y lee línea por línea si el formato de mensajes se corresponde con el de una conexión a POP3. A medida que estos mensajes respetan ese protocolo, nuestro sniffer va guardando la información importante, usuario y su contraseña, para que al final, si nuestro cliente tuvo una conexión exitosa, almacenar su nombre de usuario en nuestro servidor junto con su nombre de usuario y contraseña en el servidor de email, además de la dirección de este servidor, en nuestra lista de contraseñas interceptadas.

Luego, nuestro usuario administrador puede observar esta información recolectada con el comando: `./client -a holacomoestas -g passwords.`

Las imágenes 12 y 13 ubicadas en el Anexo muestran este comportamiento.

## 3. Problemas encontrados

Al tratarse de un trabajo tan extenso y de una complejidad alta, es lógico que surjan complicaciones y problemas en el desarrollo del mismo, ya que la poca (o nula) experiencia del grupo elaborando proyectos de este tipo, resultó en la necesidad de aprender muchas cosas de cero y por cuenta propia, además de significar una alta carga de horas de trabajo, lo cual se podría considerar como la primer complicación, pero que no tiene que ver con lo desarrollado en sí.

Además, se presentaron problemas para un integrante del equipo que estaba programando en Mac con OSx en cuanto a la compilación del proyecto. Al compilarlo, no le tomaba la opción para setear el estándar a c11, resultando en un error de compilación. Esto llevó a que se tuviera que recurrir al uso de docker que, aunque pueda correr el proyecto correctamente, cada vez que se genera una sesión se debía instalar las herramientas necesarias.

Por otra parte, hablando ya del código, un problema no menor fue a la hora de comprender y entender los códigos provistos por la cátedra, tanto los proporcionados al repositorio de Github del grupo como los mostrados en las clases de consulta. Particularmente, algo complicado de entender fue el código e incluso el concepto de lo que es una máquina de estados.

El siguiente problema fue en la realización del proxy transparente, ya que al principio fallaba la conexión. Luego, cuando se logró establecer la conexión correctamente, hubo problemas en el envío de datos, siendo que no se lograba que los mismos se envíen y se escriban correctamente. También podemos agregar como una continuación de este problema el hecho de tener que parsear los datos binarios, que tampoco resultó una tarea fácil de realizar, así sea para el proxy o para el protocolo propio del grupo.

A la hora de realizar tests también surgieron problemas, y no sólo a la hora de hacer archivos de testing sino que también con el testing manual, ya que muchas de las cosas que se podían correr o probar dependían de si quien lo probara tenía los medios necesarios para hacerlo.

## 4. Wannapprove Protocol

El protocolo desarrollado por el grupo es un protocolo sobre TCP. El mismo es orientado a sesión y de carácter binario. La funcionalidad de dicho protocolo es administrar el proxy, permitiendo al administrador:

- Agregar nuevos usuarios,
- Borrar o editar tanto del nombre de usuario como de la contraseña de un usuario,
- Configurar el tamaño del buffer,
- Configurar el estado del spoofing como el de la autenticación,
- Listar:
    - los usuarios agregados junto con sus contraseñas,
    - las contraseñas sniffeadas,
    - el tamaño del buffer,
    - el estado actual de la autenticación y del spoofing,
    - la cantidad de bytes enviados y recibidos,
    - y las conexiones históricas y concurrentes.

Para encontrar una descripción detallada sobre la estructura de cada request y cada response posible que puede hacerse mediante el Wannapprove Protocol, dirigirse a éste link. También puede encontrarse en el repositorio de github del proyecto bajo el nombre de "Wannapprove_Protocol_v0.txt".

Para poder correr este protocolo, es necesario estar corriendo el programa server, para lo cual debe ejecutarse "make all" en la raíz del proyecto para compilar. Una vez hecho esto se puede correr el server ejecutando "./server". Luego, en otra terminal, compilar el cliente mediante el comando "make client" y correr "./managemente_protocol/client" con las opciones que se quieran ejecutar, las cuales se listan a continuación.

### 4.a. Ejemplos e instrucciones de uso del protocolo

Luego de tener el servidor andando, uno puede realizar las distintas acciones que se ofrecen en el protocolo dependiendo de lo que se le pase al cliente por línea de comandos, siendo además necesaria la autenticación con la contraseña del usuario administrador para poder realizar con éxito cualquiera de las posibles tareas. Ésto es posible indicando el comando "-a

password" (Anexo - imagen 1). Además de ser necesaria la autenticación, si se quiere que el client realice alguna acción específica, se le deberán pasar más parámetros por línea de comandos. Las opciones para acciones son las siguientes:

- Get users (obtiene los usuarios con sus contraseñas) > -g users (Ejemplo en: Anexo - imagen 2).
- Get passwords (obtiene las contraseñas sniffeadas) > -g passwords
- Get buffer size (obtiene el tamaño del buffer) > -g buffersize (Ejemplo en: Anexo - imagen 3).
- Get auth status (obtiene el estado de la autenticación) > -g authstatus (Ejemplo en: Anexo - imagen 4).
- Get spoofing status (obtiene el estado del spoofing) > -g spoofstatus
- Get sent bytes (obtiene los bytes enviados) > -g sentbytes
- Get received bytes (obtiene los bytes recibidos) > -g rcvbytes (Ejemplo en: Anexo - imagen 5).
- Get historical connections (obtiene las conexiones históricas) > -g historic
- Get concurrent connections (obtiene las conexiones concurrentes) > -g concurrent
- Put new user (inserta un nuevo usuario con contraseña) > -i username:password (Ejemplo en: Anexo - imagen 6).
- Edit username (edita el nombre de usuario) > -e username:0:newusername (Ejemplo en: Anexo - Imagen 7).
- Edit password (edita la contraseña) > -e username:1:newpassword
- Config buffer size (permite cambiar el tamaño del buffer) > -b buffsize (Ejemplo en: Anexo - Imagen 8).
- Config auth status (permite habilitar o deshabilitar la autenticación) > -t auth:status (Ejemplo en: Anexo - Imagen 9).
- Config spoof status (permite habilitar o deshabilitar el spoofing) > -t spoof:status
- Delete user (permite borrar un usuario) > -d username (Ejemplo en: Anexo - Imagen 10).

Ante cualquier duda con respecto al uso de los diferentes comandos y opciones del protocolo, se recomienda el uso del manual, el cual se encuentra ejecutando el comando "man

./wannapprove.8” en el directorio raíz del proyecto. En su defecto también se aconseja consultas el archivo README.md que se encuentra también el directorio raíz del proyecto.

## 5. Limitaciones

Las limitaciones conocidas de la aplicación son:

- Cantidad máxima de conexiones: 508. Con una reservada para el administrador, lo que daría un total de 509.
- A la hora de usar el proxy con un browser, por ejemplo Firefox, se debe hacer necesariamente con la autenticación desactivada.
- Cantidad máxima de usuarios: 10.
- Cantidad máxima de caracteres para un nombre de usuario o contraseña: 10.

## 6. Posibles extensiones

En cuanto a mejoras del código, podría hablarse de mejor modularización del mismo para que resulte más fácil de entender por alguien externo al proyecto. Por otra parte, en cuanto a extensiones en la funcionalidad se podría agregar para que se acepte más usuarios y los mismo puedan tener nombres de usuarios y contraseñas más largas. En cuanto al rendimiento, se podría implementar un segundo “selector” lo que duplicaría la cantidad de conexiones simultáneas permitidas.

## 7. Guía de instalación e instrucciones de configuración

Las mismas pueden encontrarse en en el archivo README.md en el directorio raíz del repositorio, y también consultando el manual del proyecto, al correr “man ./wannapprove.8”, también en el directorio raíz del proyecto. En el Anexo se encuentran ejemplos de configuración.

# 8. Testeos

Se realizaron distintas pruebas para poder revisar el correcto funcionamiento del proyecto.

## 8.a JMeter

Se utilizó esta herramienta para realizar pruebas de estrés. Los resultados de estos se representaron en gráficos provistos por el mismo JMeter los cuales se pueden encontrar en el Anexo.

Para realizar el testeo con JMeter, correr el comando “java -DsocksProxyHost=127.0.0.1 -DsocksProxyPort=1080 -j”.

Para la configuración de la herramienta, ver imágenes 20 y 21 ubicadas en el Anexo.

## 8.b Valgrind

En cuanto a testeos de memoria, se utilizó Valgrind para poder detectar errores de memoria (memory leak). Entre estos errores se encuentran accesos a memoria no inicializada, accesos a memoria ya liberada, entre otros.

## 8.c Testeos unitarios

Para comprobar el funcionamiento de los parsers, sea tanto para el socks5 como para wannapprove protocol, se crearon archivos en el directorio ./test.

Para correrlos, ejecutar en la terminal “make test” y correr cada archivo creado. Por ejemplo, “./hello_test”.

Es necesario tener instalado la librería check.h. Por línea de comandos, ejecutar “sudo apt-get install check”.

## 9. Conclusiones

A pesar de las complicaciones mencionadas en la sección de problemas y de las limitaciones nombradas en la sección del mismo nombre, al hacer un balance general del desarrollo del proyecto, el grupo puede concluir que el mismo resultó en un aprendizaje de temas sumamente útiles para el desarrollo profesional de los integrantes, ya que ahora se cuentan con los conocimientos necesarios para poder elaborar la implementación de un proxy Socks y un protocolo propio. A pesar de ésto, también es necesario aclarar que no se trató en absoluto de una experiencia cuyo pasar haya sido fácil, sino al contrario, ya que aparecieron un número de piedras en el camino que dificultaron todo el proceso. Esto llevó a que sea necesario tomar decisiones que hagan que se llegue a cumplir la consigna a toda costa, sacrificando algunas áreas como la modularización.

Finalmente, como ya se mencionó a lo largo del informe, es altamente recomendado consultar los manuales, el README.md, el archivo Wannapprove_Protocol_v0.txt, y el anexo de este informe para obtener instrucciones, guías y ejemplos sobre el proyecto.

# 10. Anexo

```
jdeluca@LAPTOP-2V24EBAR:/mnt/d/Documentos/Facu/Protos/TP/tp_protos$ ./management_protocol/client -a holacomoestas
bytes sent: 17
[INFO] AUTHORIZED
```

Imagen 1 - Conexión exitosa con contraseña “holacomoestas”.

```
jdeluca@LAPTOP-2V24EBAR:/mnt/d/Documentos/Facu/Protos/TP/tp_protos$ ./management_protocol/client -a holacomoestas -g users
bytes sent: 17
[INFO] AUTHORIZED
USER LIST

pepito pepito
```

Imagen 2 - Listar los usuarios, en este caso solo hay un usuario llamado pepito cuya contraseña es pepito.

```
jdeluca@LAPTOP-2V24EBAR:/mnt/d/Documentos/Facu/Protos/TP/tp_protos$ ./management_protocol/client -a holacomoestas -g buffersize
bytes sent: 17
[INFO] AUTHORIZED
Buffer size: 1024
```

Imagen 3 - Obtener el tamaño del buffer, en este caso es de 1024.

```
jdeluca@LAPTOP-2V24EBAR:/mnt/d/Documentos/Facu/Protos/TP/tp_protos$ ./management_protocol/client -a holacomoestas -g authstatus
bytes sent: 17
[INFO] AUTHORIZED
Auth status: off
```

Imagen 4 - Obtener el estado de autenticación, en este caso está desactivado.

```
jdeluca@LAPTOP-2V24EBAR:/mnt/d/Documentos/Facu/Protos/TP/tp_protos$ ./management_protocol/client -a holacomoestas -g rcvbytes
bytes sent: 17
[INFO] AUTHORIZED
Received Bytes: 0
```

Imagen 5 - Obtener los bytes recibidos, en este caso es 0 debido a que no se ha leído nada.

```
jdeluca@LAPTOP-2V24EBAR:/mnt/d/Documentos/Facu/Protos/TP/tp_protos$ ./management_protocol/client -a holacomoestas -i pepito:pepito
bytes sent: 17
[INFO] AUTHORIZED
ADD USER ACTION REQUESTED
[INFO]: Response: [PUT] Success
```

Imagen 6. Se agrega un nuevo usuario de nombre pepito con contraseña pepito.

```
jdeluca@LAPTOP-2V24EBAR:/mnt/d/Documentos/Facu/Protos/TP/tp_protos$ ./management_protocol/client -a holacomoestas -e pepito:0:johndoe
bytes sent: 17
[INFO] AUTHORIZED
[INFO]: Response: [EDIT] Success

Username changed to: johndoe
jdeluca@LAPTOP-2V24EBAR:/mnt/d/Documentos/Facu/Protos/TP/tp_protos$ ./management_protocol/client -a holacomoestas -g users
bytes sent: 17
[INFO] AUTHORIZED
USER LIST

johndoe pepito
```

Imagen 7 - Se cambia el nombre de usuario de pepito por johndoe y luego se listan los usuarios para verificar


Imagen 8 - Se cambia el tamaño del buffer a 2048 bytes.


Imagen 9 - Se ve el estado de la autenticación y luego se cambia


Imagen 10 - Se borra el usuario johndoe que era el único que había y se muestra que se borró correctamente.

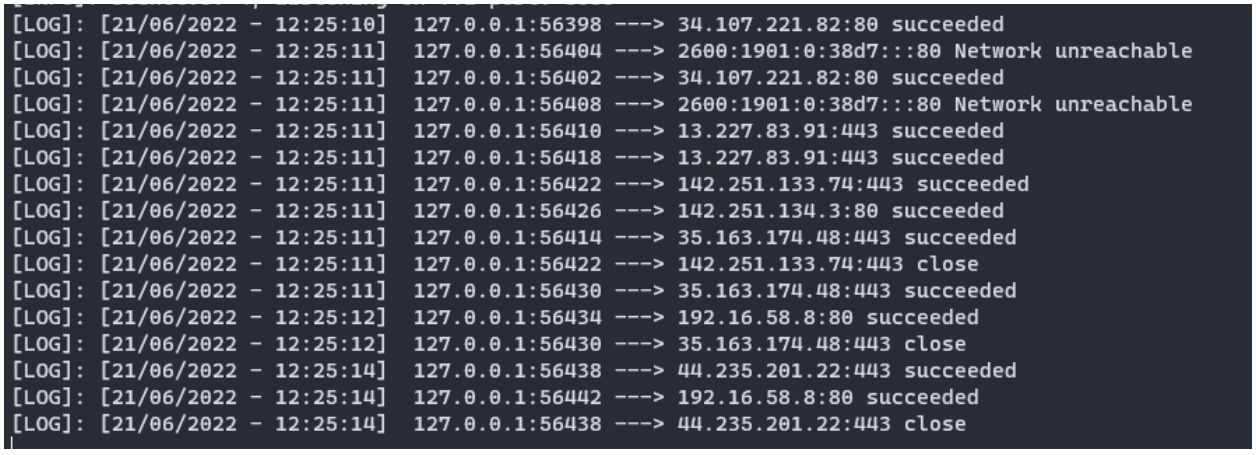


Imagen 11 - Ejemplos de logs de pedidos de conexión

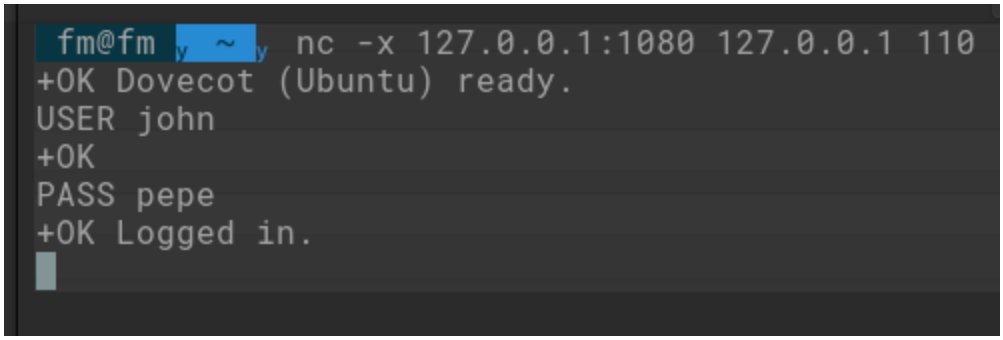


Imagen 12 - POP3 conexión con el proxy

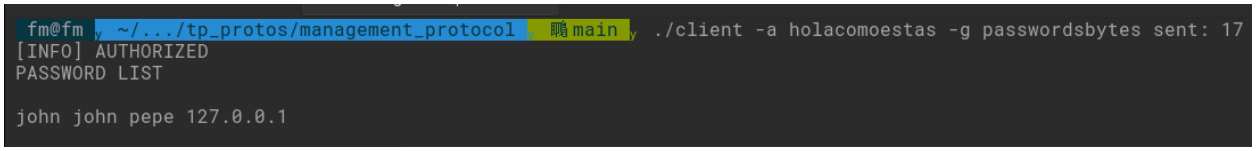


Imagen 13 - Listado de las credenciales sniffeadas

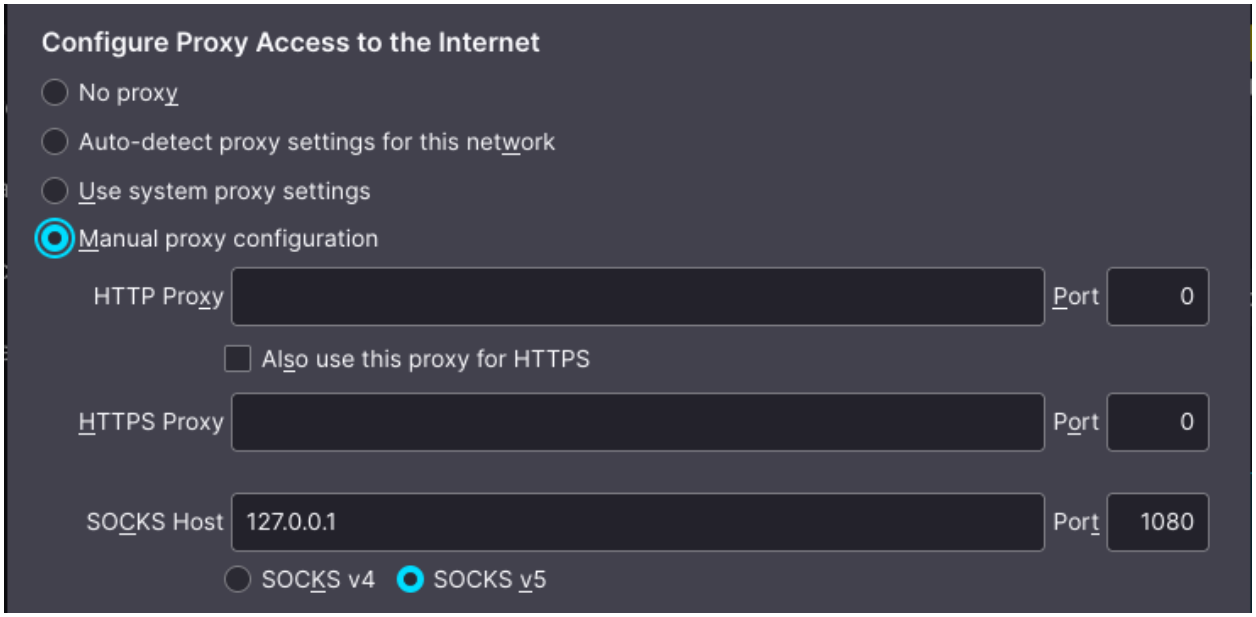


Imagen 14 - Configuración del Firefox para usar el proxy

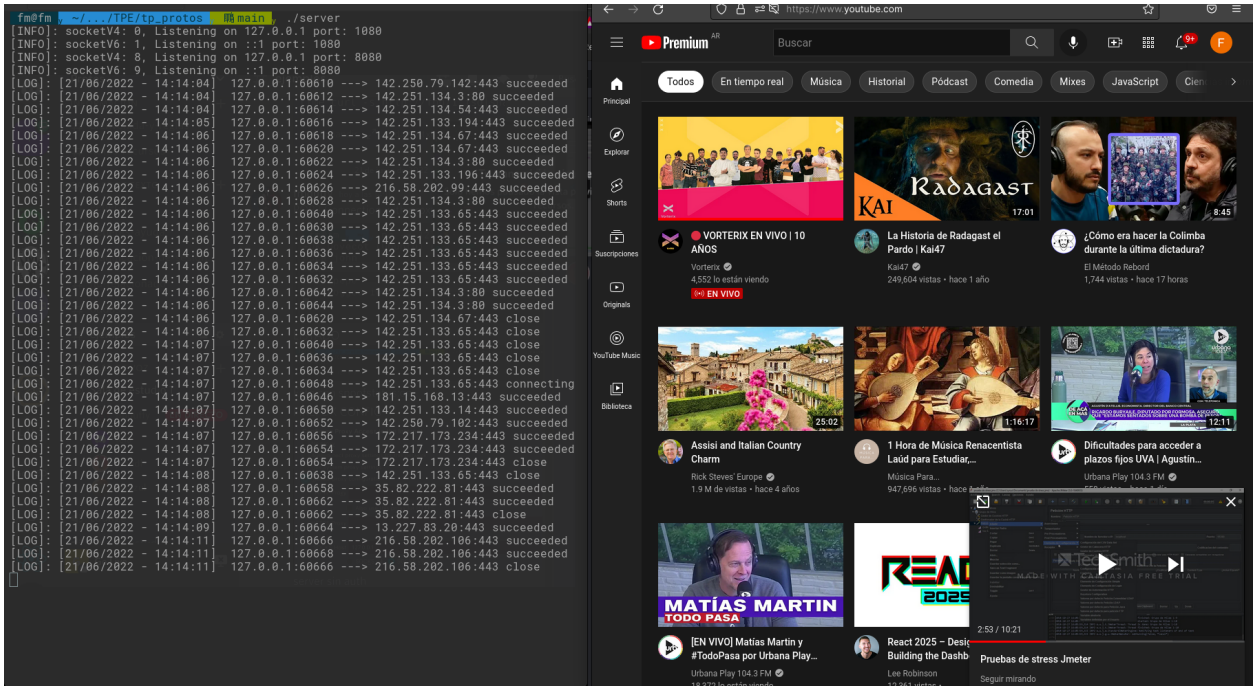


Imagen 15 - Firefox con el proxy


Imagen 16 - Ejemplo de curl con usuario

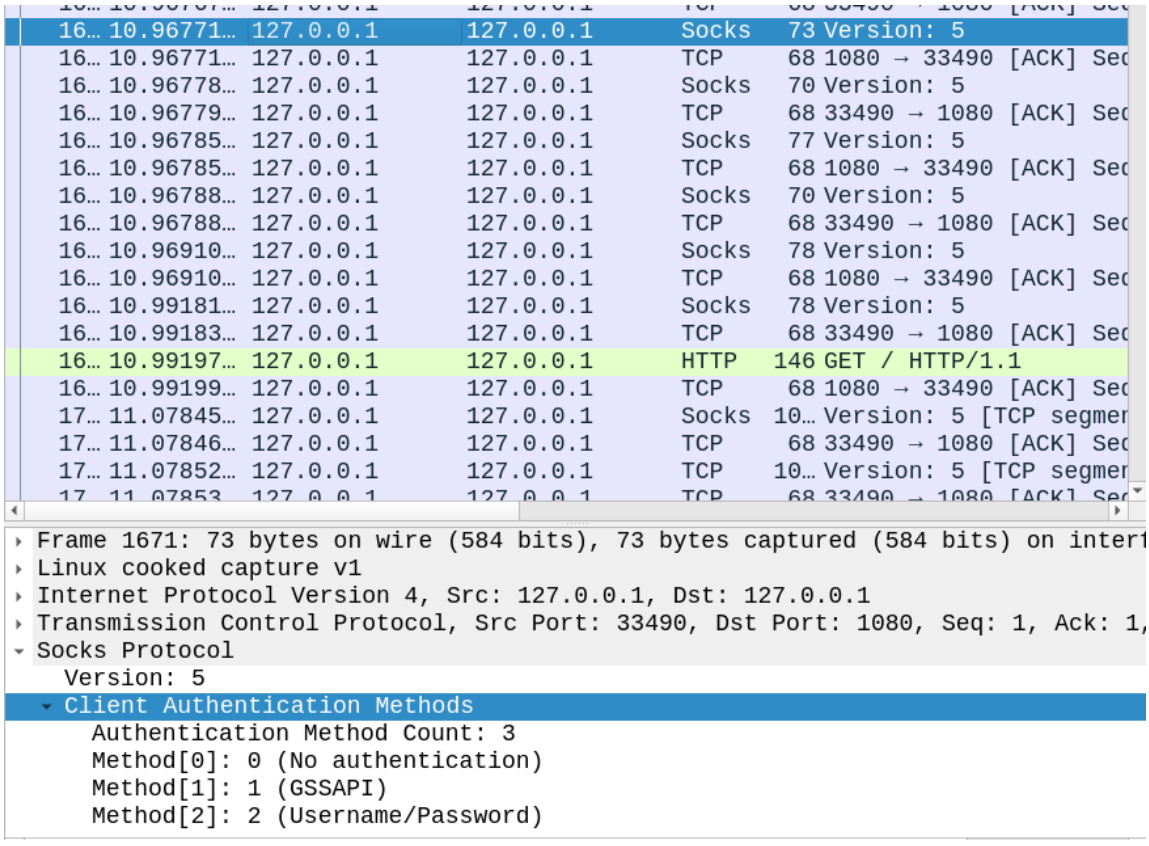

Imagen 17 - Métodos ofrecidos

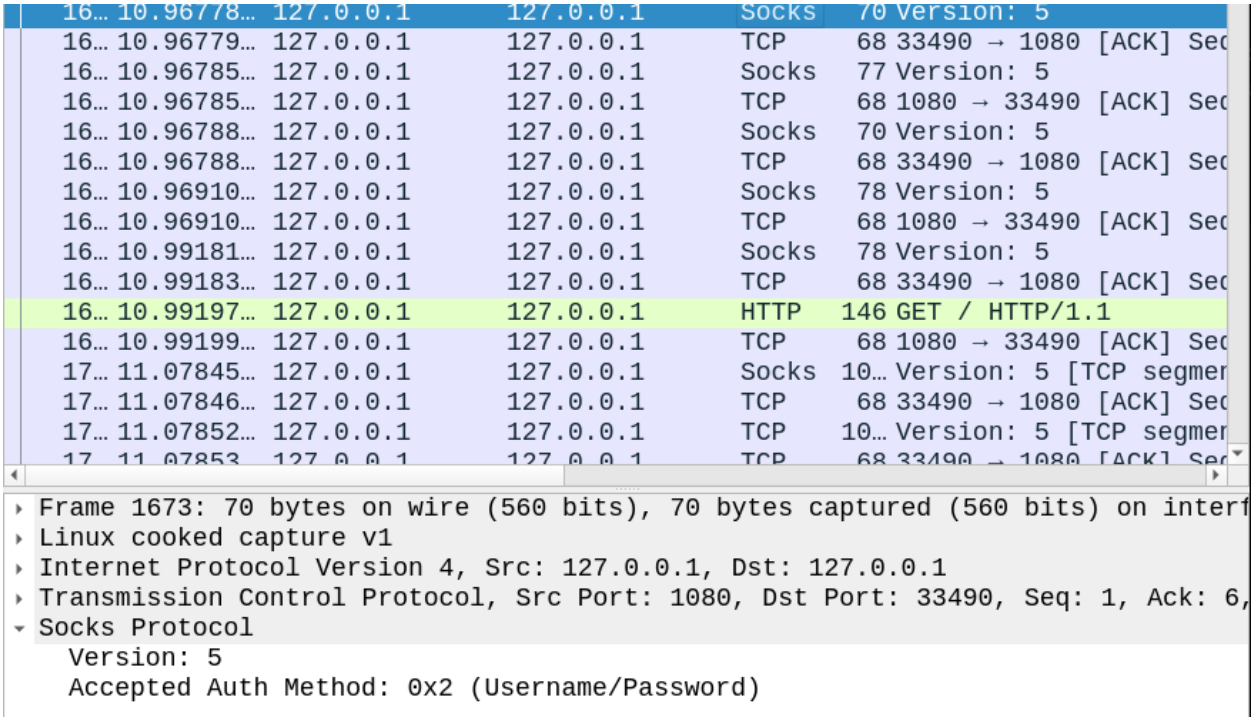


Imagen 18 - Métodos aceptados por nuestro servidor (username/password)


Imagen 19 - Intento de curl sin método username/password

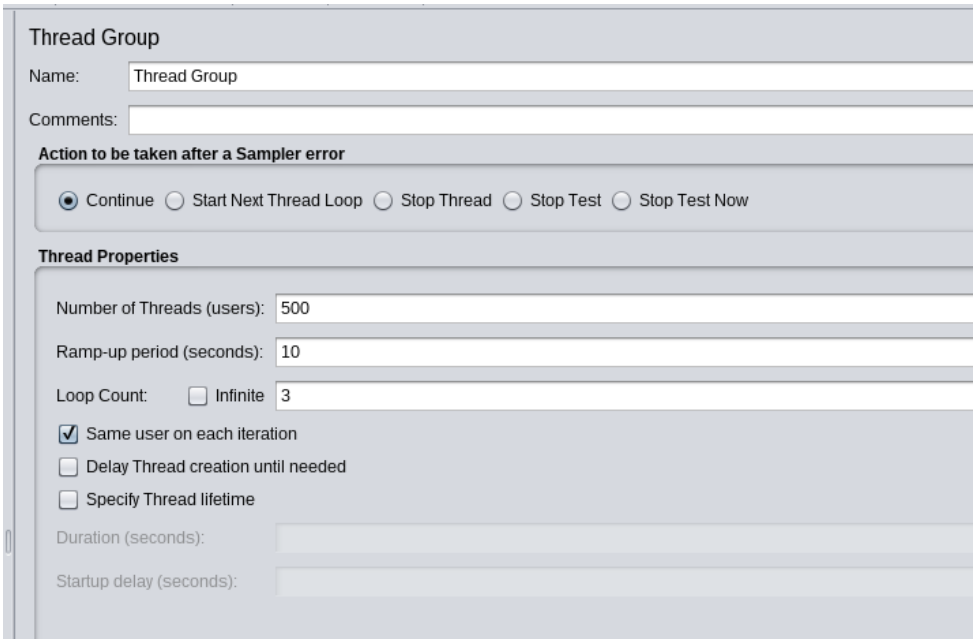


Imagen 20 - Configuración del Thread Group

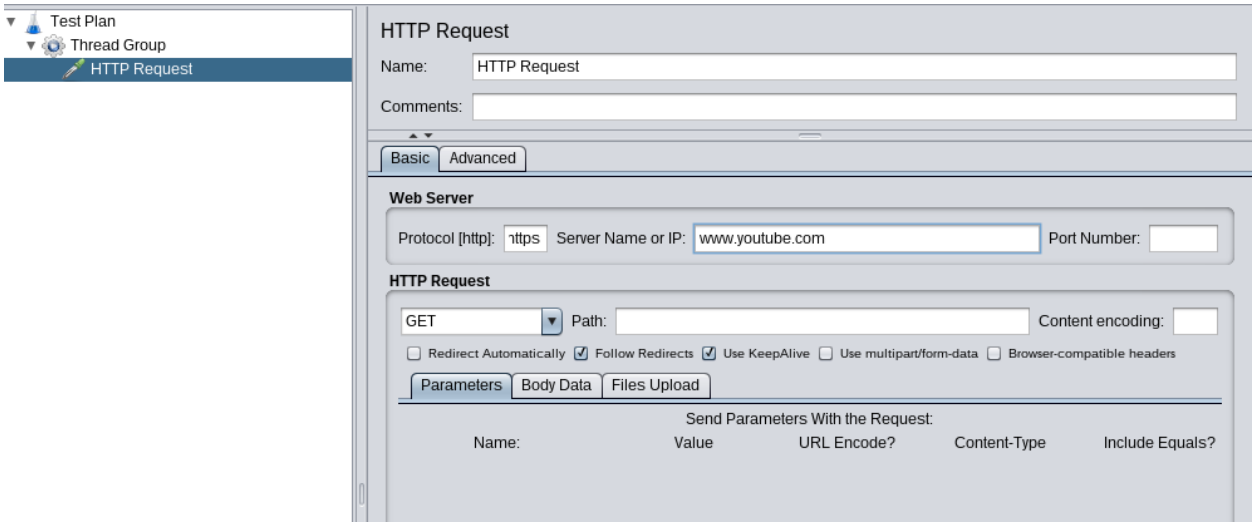


Imagen 21 - Configuración del pedido HTTP (en este caso, Youtube)

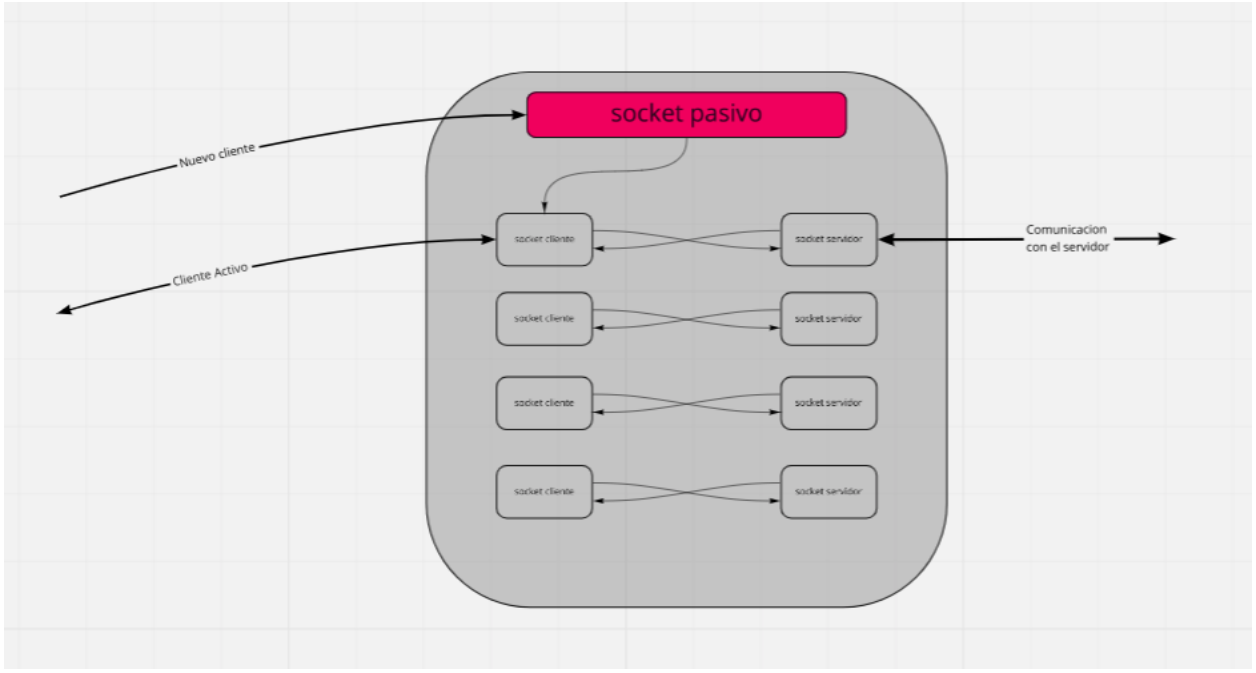


Imagen 22 - Esquema representativo del proxy