Trabajo Práctico Final
Protocolos de comunicación (72.07)

Primer cuatrimestre de 2025

Grupo 12

Integrantes:

Juan Bautista Albertoni Salini	**64189**
Santiago Devesa	**64223**
Tiziano Fuchinecco	**64191**
Junior Rambau	**64461**

Fecha de entrega: Martes 15 de julio 2025

ÍNDICE

Descripción detallada de los protocolos y aplicaciones desarrolladas

Protocolo de Autenticación y Administración

Protocolo de Autenticación

Nuestro protocolo de autenticación, basado en el estándar definido en la RFC 1929, garantiza que solo los usuarios autorizados y autenticados puedan acceder a las funcionalidades administrativas de nuestro servidor

Formato de Mensaje de Autenticación (Cliente -> servidor)

VERSION	USER-LENGTH	USER	PASS-LENGTH	PASS
1 Byte	1 Byte	N Bytes	1 Byte	N Bytes

- Versión: Versión del protocolo (0x01)
- User-Length: Longitud del nombre de usuario
- User: Nombre de usuario
- Pass-Length: Longitud de la contraseña
- Pass: contraseña

Formato de la respuesta (Servidor -> Cliente)

VERSION	STATUS
1 Byte	1 Byte

- Versión: Versión del protocolo (0x01)
- Status: Codigo de estado (0x0 éxito, otro fallo)

Flujo de funcionamiento

1. El cliente establece una conexión TCP con el servidor.
2. Una vez conectado, el cliente envía el mensaje de autenticación siguiendo el formato antes mencionado.
3. El servidor procesa este mensaje mediante un parser específico para llevar a cabo la autenticación (*management_auth*), validando las credenciales contra una base de datos. En caso de que las credenciales sean válidas, el servidor responde con un código de éxito y habilita al cliente a enviar comandos administrativos. En caso contrario, se responde con un código de fallo.

Protocolo de Administración

En cuanto a los usuarios, se tomó la decisión de clasificarlos en dos categorías: Administradores y usuarios estándar. Los administradores tienen acceso total a todos los comandos disponibles, mientras que los usuarios estándar tienen permisos más limitados.

Una vez autenticado por el servidor, el cliente puede enviar comandos para gestionar usuarios, consultar estadísticas y realizar otros permitidos en base a sus privilegios.

Formato de Mensaje (Cliente -> servidor)

VERSION	COMMAND	PAYLOAD-LEN	PAYLOAD
1 Byte	1 Byte	1 Byte	N Bytes

- Versión: Versión del protocolo (0x01)
- Command: Identificador del comando
- Payload-Len: Longitud del payload
- Payload: Datos adicionales

El payload incluye los argumentos del comando. Dichos argumentos deben estar delimitados por el carácter ‘$’ en caso de ser necesario.

Formato de la respuesta (Servidor -> Cliente)

VERSION	STATUS	MESSAGE
1 Byte	1 Byte	N Bytes

- Versión: Versión del protocolo (0x01)
- Status: Codigo de estado (0x0 éxito, otro fallo)
- Message (Opcional): Mensaje sobre la respuesta

Comandos Implementados

USERS

Lista todos los usuarios registrados en el sistema

```
jrambau@jrambau:/Rproxy$ ./bin/client $ $ admin 1234 USERS
```

Para listar los usuarios, el cliente envía esto a modo de request:

VERSION	COMMAND	PAYLOAD_LEN
0x01	0x01	0x00

- Versión: Versión del protocolo (0x01)
- Command: El byte 0 correspondiente al comando USERS
- Payload_len: 0x00 ya que no hay argumentos para este comando
- Payload: Vacío

En caso de que se pueda mostrar correctamente los usuarios, el servidor envía lo siguiente a modo de respuesta:

VERSION	STATUS	MESSAGE
0x01	0x00	“users count: X\n<user1>\n<user2>\n…\0”

- Versión: Versión del protocolo (0x01)
- Status: 0x00 correspondiente a éxito
- Message: Devuelve la cantidad de usuarios registrados y sus nombres. Todos separados por saltos de línea

En caso de que no se puedan mostrar los usuarios, el servidor puede enviar lo siguiente a modo de respuesta:

VERSION	STATUS	MESSAGE
0x01	0x04	“users: invalid argument count, expected 0\0”

- Versión: Versión del protocolo (0x01)
- Status: 0x04 correspondiente a que se enviaron argumentos inválidos para este comando
- Message: Devuelve la cantidad de usuarios registrados y sus nombres. Todos separados por saltos de línea

ADD_USER

Agrega un usuario al servidor. Para agregarlo debe indicar el nombre del usuario y la contraseña.

```
jrambau@jrambau:/Rproxy$ ./bin/client $ $ admin 1234 ADD_USER jrambau 1234
```

Para agregar usuarios, el cliente envía esto a modo de request:

VERSION	COMMAND	PAYLOAD_LEN	PAYLOAD
0x01	0x01	0x0C	“jrambau:1234”

- Versión: Versión del protocolo (0x01)
- Command: El byte 1 correspondiente al comando ADD_USER
- Payload_len: 13 bytes correspondientes a la longitud del payload
- Payload: Nombre de usuario y contraseña separados por :

En caso de que el usuario se agregue correctamente, el servidor envía lo siguiente a modo de respuesta:

VERSION	STATUS	MESSAGE
0x01	0x00	“add_user: user added successfully\0”

- Versión: Versión del protocolo (0x01)
- Status: 0x00 correspondiente a que se agregó exitosamente
- Message: Mensaje explicativo enviado por el servidor

En caso de que no se pueda agregar al usuario, el servidor puede enviar lo siguiente a modo de respuesta:

VERSION	STATUS	MESSAGE
0X01	0X01	“command requires admin privileges\0”
0x01	0x04	“add_user: invalid argument count, expected 2\0”
0x01	0x05	“add_user: user already exists\0”
0x01	0x09	“add_user: failed to add user\0”

- Versión: Versión del protocolo (0x01)
- Status: 0x00 correspondiente a que se agregó exitosamente
    - 0x01 correspondiente a que no se tiene los permisos necesarios
    - 0x04 correspondiente a que es inválida la cantidad de argumentos
    - 0x05 correspondiente a que el usuario ya existe
    - 0x09 correspondiente a que hubo un error en el servidor
- Message: Mensaje explicativo enviado por el servidor

DELETE_USER

Elimina un usuario del servidor. Para eliminarlo debe indicar el nombre de usuario

```
jrambau@jrambau:/Rproxy$ ./bin/client $ $ admin 1234 DELETE_USER jrambau
```

VERSION	COMMAND	PAYLOAD_LEN	PAYLOAD
0x01	0x02	0x07	“jrambau”

- Versión: Versión del protocolo (0x01)
- Command: El byte 2 correspondiente al comando DELETE_USER
- Payload_len: 7 bytes correspondientes a la longitud del payload
- Payload: Nombre de usuario a eliminar

En caso de que el usuario se elimine correctamente, el servidor envía lo siguiente a modo de respuesta:

VERSION	STATUS	MESSAGE
0x01	0x00	“delete_user: user deleted successfully\0”

- Versión: Versión del protocolo (0x01)
- Status: 0x00 correspondiente a que se agregó exitosamente
- Message: Mensaje explicativo enviado por el servidor

En caso de que no se pueda eliminar al usuario, el servidor puede enviar lo siguiente a modo de respuesta:

VERSION	STATUS	MESSAGE
0X01	0X01	“command requires admin privileges\0”
0x01	0x04	“delete_user: invalid argument count, expected 2\0”

0x01	0x06	“delete_user: user not found\0”
0x01	0x09	“delete_user: failed to delete user\0”

- Versión: Versión del protocolo (0x01)
- Status:
  - 0x01 correspondiente a que no se tiene los permisos necesarios
  - 0x04 correspondiente a que es inválida la cantidad de argumentos
  - 0x06 correspondiente a que el usuario no existe
  - 0x09 correspondiente a que hubo un error en el servidor
- Message: Mensaje explicativo enviado por el servidor

CHANGE_PASSWORD

Cambia la contraseña de un usuario. Para cambiarla debe indicar el nombre de usuario y la nueva contraseña

```
jrambau@jrambau:/Rproxy$ ./bin/client $ $ admin 1234 CHANGE_PASSWORD jrambau 1234
```

Para cambiar contraseñas, el cliente envía esto a modo de request:

VERSION	COMMAND	PAYLOAD_LEN	PAYLOAD
0x01	0x03	0x0C	“jrambau:1234”

- Versión: Versión del protocolo (0x01)
- Command: El byte 3 correspondiente al comando CHANGE_PASSWORD
- Payload_len: 12 bytes correspondientes a la longitud del payload
- Payload: Nombre de usuario y contraseña separados por :

En caso de que la contraseña se cambie correctamente, el servidor envía lo siguiente a modo de respuesta:

VERSION	STATUS	MESSAGE
0x01	0x00	“change_password: password changed successfully\0”

- Versión: Versión del protocolo (0x01)
- Status: 0x00 correspondiente a que se cambió exitosamente
- Message: Mensaje explicativo enviado por el servidor

En caso de que no se pueda eliminar al usuario, el servidor puede enviar lo siguiente a modo de respuesta:

VERSION	STATUS	MESSAGE
0X01	0X01	“command requires admin privileges\0”
0x01	0x04	“change_password: invalid argument count, expected 2\0”
0x01	0x06	“change_password: user not found\0”
0x01	0x09	“change_password: failed to change password\0”

- Versión: Versión del protocolo (0x01)
- Status:
  - 0x01 correspondiente a que no se tiene los permisos necesarios
  - 0x04 correspondiente a que es inválida la cantidad de argumentos
  - 0x06 correspondiente a que el usuario no existe
  - 0x09 correspondiente a que hubo un error en el servidor
- Message: Mensaje explicativo enviado por el servidor

STATS

Muestra las estadísticas del servidor

```
jrambau@jrambau:/Rproxy$ ./bin/client $ $ admin 1234 STATS
```

Para mostrar las estadísticas, el cliente envía esto a modo de request:

VERSION	COMMAND	PAYLOAD_LEN
0x01	0x04	0x00

- Versión: Versión del protocolo (0x01)
- Command: El byte 4 correspondiente al comando STATS
- Payload_len: 0x00 ya que no hay argumentos para este comando
- Payload: Vacío

En caso de que se puedan mostrar las estadísticas correctamente, el servidor envía lo siguiente a modo de respuesta:

VERSION	STATUS	MESSAGE
0x01	0x00	"Current connections: 2\nTotal connections: 5\nBytes sent: 1024\nBytes received: 2048\nDNS requests: 10\nServer uptime: 0 days, 1 hours, 4 minutes, 32 seconds\n\0"

- Versión: Versión del protocolo (0x01)
- Status: 0x00 correspondiente a éxito
- Message: Mensaje explicativo enviado por el servidor, con formato legible al usuario

En caso de que no se puedan mostrar las estadísticas correctamente, el servidor envía lo siguiente a modo de respuesta:

VERSION	STATUS	MESSAGE
0x01	0x04	"stats: invalid argument count, expected 0\0"

- Versión: Versión del protocolo (0x01)
- Status: 0x04 correspondiente a que se enviaron argumentos inválidos para este comando
- Message: Mensaje explicativo enviado por el servidor

CHANGE_ROLE

Cambia el rol de un usuario. Para cambiarlo debe indicar el nombre de usuario y el rol (admin/user)

```
jrambau@jrambau:/Rproxy$ ./bin/client $ $ admin 1234 CHANGE_ROLE jrambau user
```

Para cambiar el rol, el cliente envía esto a modo de request:

VERSION	COMMAND	PAYLOAD_LEN	PAYLOAD
0x01	0x05	0x0C	“jrambau:user”

- Versión: Versión del protocolo (0x01)
- Command: El byte 5 correspondiente al comando CHANGE_PASSWORD
- Payload_len: 12 bytes correspondientes a la longitud del payload
- Payload: Nombre de usuario y contraseña separados por :

En caso de que el usuario se elimine correctamente, el servidor envía lo siguiente a modo de respuesta:

VERSION	STATUS	MESSAGE
0x01	0x00	“change_role: role changed successfully\0”

- Versión: Versión del protocolo (0x01)
- Status: 0x00 correspondiente a que se agregó exitosamente
- Message: Mensaje explicativo enviado por el servidor

En caso de que no se pueda cambiar el rol, el servidor puede enviar lo siguiente a modo de respuesta:

VERSION	STATUS	MESSAGE
0X01	0X01	“command requires admin privileges\0”
0x01	0x04	“change_role: invalid argument count, expected 2\0”
0x01	0x06	“change_role: user not found\0”
0x01	0x07	“change_role: invalid role, expected ‘admin’ or ‘user’\0”
0X01	0x09	“change_role: failed to change role\0”

- Versión: Versión del protocolo (0x01)
- Status:
    - 0x01 correspondiente a que no se tiene los permisos necesarios
    - 0x04 correspondiente a que es inválida la cantidad de argumentos
    - 0x06 correspondiente a que el usuario no existe
    - 0x07 correspondiente a que el rol es invalido
    - 0x09 correspondiente a que hubo un error en el servidor
- Message: Mensaje explicativo enviado por el servidor

SET_DEFAULT_AUTH_METHOD

Define si se necesita autenticación o no para ejecutar los comandos

```
jrambau@jrambau:/Rproxy$ ./bin/client $ $ admin 1234 SET_DEFAULT_AUTH_METHOD
no_auth
```

Para definir el nivel de autenticación, el cliente envía esto a modo de request:

VERSION	COMMAND	PAYLOAD_LEN	PAYLOAD
0x01	0x06	0X07 o 0x10	“no_auth” o “username_password”

- Versión: Versión del protocolo (0x01)
- Command: El byte 6 correspondiente al comando SET_DEFAULT_AUTH_METHOD
- Payload_len: 7 bytes correspondientes a la longitud de “no_auth” o 16 bytes correspondientes a la longitud de “username_password”
- Payload: “no_auth” o “username_password”

En caso de que se defina correctamente, el servidor envía lo siguiente a modo de respuesta:

VERSION	STATUS	MESSAGE
0x01	0x00	“set_default_auth_method: default auth method set successfully\0”

- Versión: Versión del protocolo (0x01)
- Status: 0x00 correspondiente a que se seteo exitosamente
- Message: Mensaje explicativo enviado por el servidor

En caso de que no se pueda definir correctamente, el servidor puede enviar lo siguiente a modo de respuesta:

VERSION	STATUS	MESSAGE
0X01	0X01	“command requires admin privileges\0”
0x01	0x04	“set_default_auth_method: invalid argument count, expected 1\0”
0x01	0x04	“set_default_auth_method: invalid method, expected ‘no_auth’ or ‘username_password’\0”
0X01	0x09	“set_default_auth_method: failed to set default auth method\0”

- Versión: Versión del protocolo (0x01)
- Status:
    - 0x01 correspondiente a que no se tiene los permisos necesarios
    - 0x04 correspondiente a que es inválida la cantidad de argumentos
    - 0x04 correspondiente a que el metodo es invalido
    - 0x09 correspondiente a que hubo un error en el servidor
- Message: Mensaje explicativo enviado por el servidor

GET_DEFAULT_AUTH_METHOD

Devuelve el tipo de autenticación necesario para correr las funciones.

```
jrambau@jrambau:/Rproxy$ ./bin/client $ $ admin 1234 GET_DEFAULT_AUTH_METHOD
```

Para ver el método de autenticación, el cliente envía esto a modo de request:

VERSION	COMMAND	PAYLOAD_LEN
0x01	0x07	0x00

- Versión: Versión del protocolo (0x01)
- Command: El byte 7 correspondiente al comando GET_DEFAULT_AUTH_METHOD
- Payload_len: 0x00 ya que no hay argumentos para este comando
- Payload: Vacío

En caso de que se pueda mostrar correctamente, el servidor envía lo siguiente a modo de respuesta:

VERSION	STATUS	MESSAGE
0x01	0x00	“Default auth method: no_auth\0”
0x01	0x00	“Default auth method: username_password\0”

- Versión: Versión del protocolo (0x01)
- Status: 0x00 correspondiente a que devolvió exitosamente
- Message: Mensaje explicativo enviado por el servidor

En caso de que no se pueda mostrar correctamente, el servidor puede enviar lo siguiente a modo de respuesta:

VERSION	STATUS	MESSAGE
0X01	0X01	“command requires admin privileges\0”
0x01	0x04	“get_default_auth_method: invalid argument count, expected 0’\0”
0X01	0x09	“get_default_auth_method: unknown auth method\0”

- Versión: Versión del protocolo (0x01)
- Status:
    - 0x01 correspondiente a que no se tiene los permisos necesarios
    - 0x04 correspondiente a que es inválida la cantidad de argumentos
    - 0x09 correspondiente a que hubo un error en el servidor
- Message: Mensaje explicativo enviado por el servidor

VIEW_ACTIVITY_LOG

Muestra el historial de accesos de un usuario específico

```
jrambau@jrambau:/Rproxy$ ./bin/client $ $ admin 1234 VIEW_ACTIVITY_LOG
jrambau
```

Para ver el historial, el cliente envía esto a modo de request:

VERSION	COMMAND	PAYLOAD_LEN	PAYLOAD
0x01	0x08	0x07	“jrambau”

- Versión: Versión del protocolo (0x01)
- Command: El byte 8 correspondiente al comando VIEW_ACTIVITY_LOG
- Payload_len: 7 bytes correspondientes a la longitud del payload
- Payload: Correspondiente al nombre de usuario

En caso de que se pueda mostrar, el servidor envía lo siguiente a modo de respuesta:

VERSION	STATUS	MESSAGE
0x01	0x00	"Historial de accesos para jrambau (2 registros):\n2024-06-01 10:00:00 - google.com\n2024-06-01 12:00:00 - example.org\n"

- Versión: Versión del protocolo (0x01)
- Status: 0x00 correspondiente a que devolvió exitosamente
- Message: Mensaje explicativo enviado por el servidor

En caso de que no se pueda mostrar correctamente, el servidor puede enviar lo siguiente a modo de respuesta:

VERSION	STATUS	MESSAGE
0X01	0X01	“command requires admin privileges\0”
0x01	0x04	"view_activity_log: invalid argument count, expected 1\0"
0x01	0x06	"view_activity_log: user not found\0"
0X01	0x09	“view_activity_log: No activity logs found for user\0”

- Versión: Versión del protocolo (0x01)
- Status:
    - 0x01 correspondiente a que no se tiene los permisos necesarios
    - 0x04 correspondiente a que es inválida la cantidad de argumentos
    - 0x06 correspondiente a que no se encuentra el usuario
    - 0x09 correspondiente a que no se encontraron logs
- Message: Mensaje explicativo enviado por el servidor

Dados los comandos antes mencionados, decidimos que los únicos que sean accesibles para ambos tipos de usuarios (Admin y User) sean los comandos *USERS* y *STATS* con la

finalidad de mantener una clara segmentación de privilegios y asegurar que ciertas acciones críticas sólo pueden ser ejecutadas por administradores.

Implementación Técnica

El protocolo fue implementado usando dos parsers: *management_auth* que se encarga de procesar los mensajes de autenticación y *management_command_parser*, que se encarga de interpretar y validar los diferentes comandos. Ambos parsers fueron designados siguiendo el mismo patrón utilizado en nuestro protocolo SOCKS5, utilizando máquinas de estado para controlar el flujo de los distintos mensajes. Dichas máquinas pertenecen al loop de eventos no bloqueantes mediante la librería *selector,* lo cual permite manejar múltiples conexiones de la mejor manera posible

Esta estructura permite mantener un flujo controlado para las conexiones administrativas permitiendo manejar correctamente tanto flujos normales como los casos de error.

Cliente de administración

Además del servidor, se desarrolló un cliente específico de administración el cual permite al administrador conectarse al proxy, autenticarse con credenciales válidas, utilizar comandos e interpretar las respuestas del servidor. Este cliente ofrece una sencilla interfaz para así poder facilitar la interacción del usuario con el sistema.

Se realiza un encapsulamiento de las funcionalidades de conexión, autenticación, envío de comandos y de recepción de respuesta mediantes abstraccion del uso de sockets. Así el cliente puede gestionar de manera transparente la serialización de los mensajes y otorgarle una clara respuesta al usuario.

Problemas encontrados durante el diseño y la implementación

Una de las principales dificultades durante el desarrollo fue adaptarse a la estructura y funcionamiento del protocolo SOCKS5, tal como se define en la especificación RFC 1928. Este protocolo se compone de distintas fases bien delimitadas: handshake, autenticación y request, cada una con su propio formato de mensaje, campos en un orden específico, longitudes fijas o variables, y códigos de respuesta esperados.

Debido a esta estructura, fue necesario implementar un parser dedicado para cada etapa, capaz de interpretar los mensajes byte a byte, validando sus campos y garantizando que se respetara estrictamente el formato definido por el protocolo.

Durante las primeras pruebas, se identificó un problema específico en la etapa de request: el servidor respondía con un mensaje mal formado que no cumplía con el formato especificado en la RFC. Esto impedía que herramientas como curl pudieran establecer correctamente una conexión a través del proxy SOCKS5, ya que la respuesta no era reconocida como válida. Este error se debió a una interpretación incorrecta de los campos requeridos en la respuesta del servidor.

Una vez identificado el problema y ajustado el parser para que generara una respuesta correcta y conforme a la RFC, la comunicación con clientes externos como curl se realizó exitosamente, completando correctamente todas las fases del protocolo.

Limitaciones de la aplicación

Falta de persistencia de usuarios

Todos los usuarios (incluyendo el administrador por defecto (admin, 1234)) existen solo en memoria durante la ejecución del servidor. Esto implica que si el servidor se reinicia, todos los cambios realizados (usuarios agregados, modificados o eliminados) se pierden. Asimismo las métricas que se guardan en base a ellos. Esto implica que cada vez que cuando se vuelva a correr el servidor el único usuario disponible sea admin.

Límite de usuarios

Se estableció un máximo de 128 usuarios registrados en el servidor, esto se debería cambiar en caso que el proyecto estuviese proyectado a gran escala.

Longitud de datos

Se estableció que el máximo de caracteres permitidos para usuario y contraseña sean 64 en cada uno.

Cifrado de credenciales

Se transmite usuario y contraseña en texto plano. Sin ningún tipo de cifrado o alguna medida de seguridad. En SOCKS5 sucede lo mismo.

Posibles extensiones

Persistencia

Se podrían guardar los usuarios, contraseñas y métricas en una base de datos para tener persistencia. Esto solucionaría nuestra limitación de perder todo ello al reiniciar el servidor.

Limitar acceso

Al tratarse de un servidor Proxy se podría limitar el acceso a ciertos dominios/IPs.

Seguridad

Se podría incorporar SSL/TLS (por ejemplo, usando OpenSSL) para asegurar que las credenciales no viajen en texto plano.

Bloqueo temporal

Implementar un sistema que bloquee temporalmente a un cliente tras varios intentos de autenticación fallidos consecutivos.

TimeOut

Implementar un timeout para desconectar a clientes del servidor luego de determinado tiempo sin actividad para evitar, por ejemplo, un ataque en el que usuarios maliciosos ocupan toda la disponibilidad del servidor indefinidamente y no dan lugar a que usuario legítimos puedan usar los servicios.

Conclusiones

A pesar de su complejidad, el trabajo práctico fue una buena oportunidad para aplicar muchos de los conceptos abordados durante el cuatrimestre. Pudimos experimentar con el desarrollo de una arquitectura cliente-servidor real, basada en el protocolo SOCKS5, e implementar en detalle sus distintas etapas, como el handshake, la autenticación y el manejo de solicitudes. Además, el agregado del protocolo de administración nos hizo investigar cómo se puede extender un sistema existente para incorporar funciones de monitoreo y gestión. El proyecto nos sirvió para adquirir conocimientos sobre programación con sockets, diseño modular, parsers por estado y manejo de múltiples conexiones mediante un selector no bloqueante.

Ejemplos de prueba

A continuación se presentan diferentes comentarios correspondientes a pruebas de estrés para el servidor.

Máxima cantidad de conexiones concurrentes:

Se realizó un script de prueba diseñado para evaluar la capacidad del servidor SOCKS5 de aceptar múltiples conexiones simultáneas. El programa intenta abrir una gran cantidad de sockets TCP hacia el servidor (sin cerrarlos), repitiendo el proceso hasta que ocurra un error en alguna conexión. Esto permitió observar cómo responde el servidor ante una carga alta de conexiones concurrentes. Se obtuvo una cantidad de **1041** conexiones simultáneas hasta recibir el error *too many open fd*

Deterioro del Throughput con respecto a la cantidad de conexiones simultáneas:

Durante esta prueba se hicieron pruebas con diferentes grupos de conexiones, empezando por 50 hasta 5000. El script inicia conexiones y si cada conexión resuelve su petición cierra, midiendo el tiempo entre que empieza la primera y termina la última. De esta forma se obtuvieron los siguientes resultados:

Conexiones	Tiempo (s)	Throughput (req/s)
10	0.51	19.61
50	1.02	49.02
100	1.45	68.97
200	2.56	78.13
500	5.79	86.35
1000	10.95	91.32
2000	21.52	92.96
5000	53.43	93.57

Tabla: Throughput en función de las conexiones.

A partir de estos resultados se pudo observar que el throughput mejora conforme aumentan las conexiones. Sin embargo, a partir de 1000 solicitudes concurrentes, el throughput ya no mejora significativamente.

Entre 1000 y 5000, se gana muy poco rendimiento por cada conexión adicional. Si bien el throughput se estabiliza, el tiempo crece linealmente.

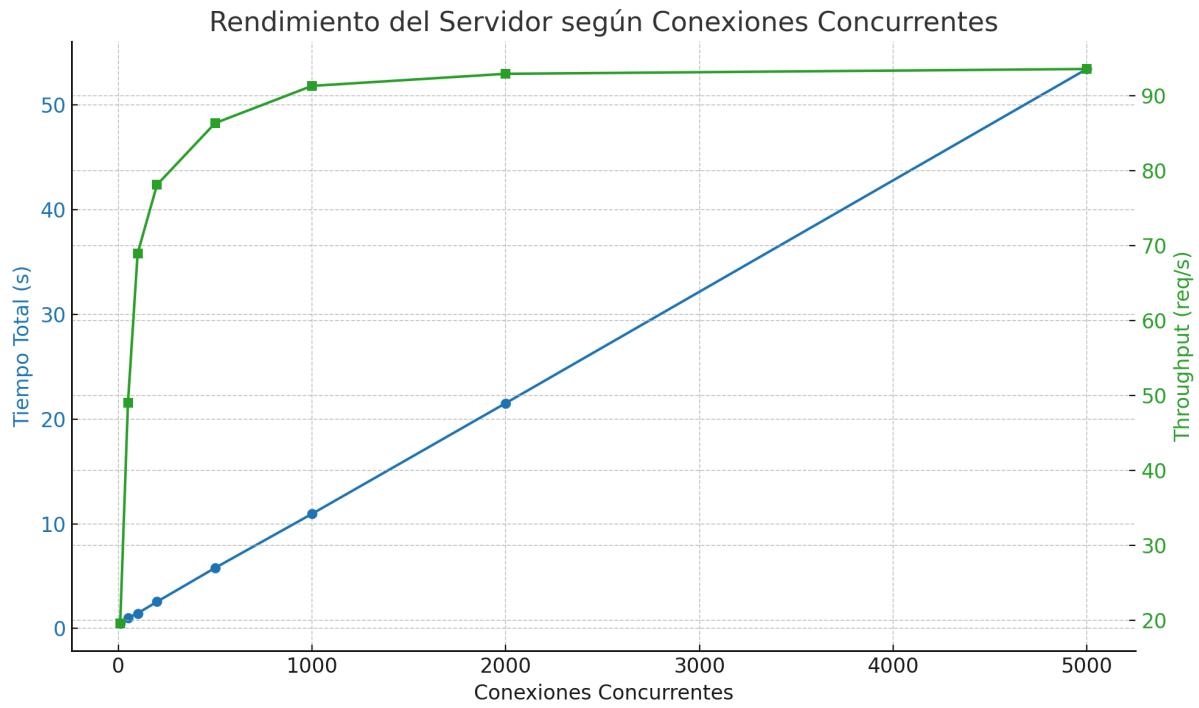

Tabla: Tiempo y throughput en función de las conexiones.

Guía de compilación

Para compilar se requiere tener instalado *gcc* y *Make,* luego desde la raíz del proyecto *(RProxy)* ejecutar el comando *"make all".* Esto generará dos ejecutables en la ubicación *RProxy/bin,* serán: *socks5v* y *client.* Luego serán utilizados para correr el servidor SOCKS5 y el cliente respectivamente.

El servidor SOCKS5 se corre desde la raíz del proyecto de la siguiente manera:

./bin/socks5v [args]

Puede correrse

./bin/socks5v -h

para ver todas las opciones disponibles.

El cliente se corre desde la raíz del proyecto de la siguiente manera:

./bin/client <host | $> <port | $> <username> <password> COMMAND

Siendo "$" la opción default de *host (127.0.0.1)* y *puerto (1080).*

Instrucciones para la configuración

Tanto para el servidor proxy SOCKS5 y para el cliente de monitoreo, la primera conexión debe hacerse con el usuario *admin* y contraseña *1234*. A partir de ahí se pueden crear usuarios propios con el comando *ADD_USER.*

Figura: Petición usando default user.

Otra alternativa es correr el servidor con el flag -u y crear un usuario para que pueda usar en el servidor proxy.

./bin/socks5v -u usuario:contraseña

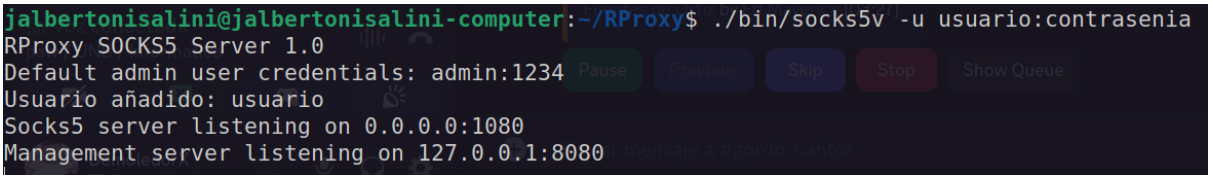

Figura: Creación de un usuario al correr el servidor.

Ahora ya puede usarse este usuario para realizar otras peticiones.

Ejemplos de configuración y monitoreo

Cliente de Monitoreo:

Conexión al servidor sin agregar parámetro “**OPTION**”

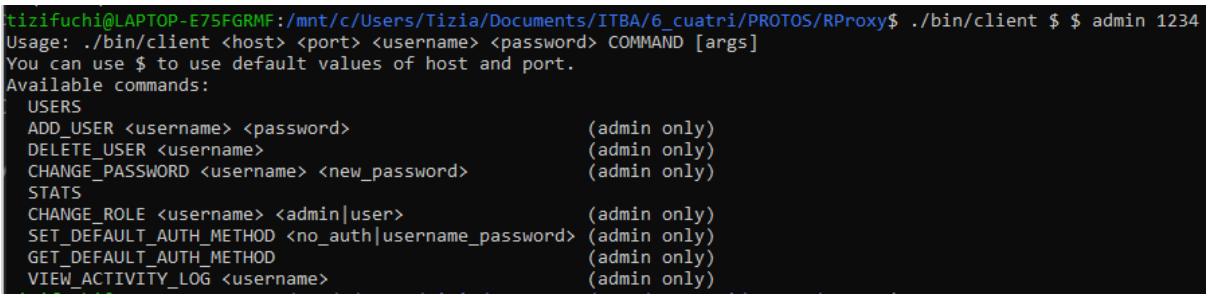

Figura 1a: Se muestra el menú con opciones posibles.

Ejecutar cualquier comando ingresando mal el nombre de usuario/contraseña

```
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$ ./bin/client $ $ admin 12123 GET_DEFAULT_AUTH_METHOD
Connecting to 127.0.0.1:8080 as admin...
Connection established with server
Auth failed
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$
```

Figura 1b: No permite autenticar

ADD_USER

Desde un usuario administrador

```
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$ ./bin/client $ $ admin 1234 ADD_USER tfuchinecco 1234
Connecting to 127.0.0.1:8080 as admin...
Connection established with server
add_user: user added successfully
```

Figura 2a: Se agregó un usuario exitosamente.

Desde un usuario simple

```
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$ ./bin/client $ $ tfuchinecco 1234 ADD_USER tfuchinecco2 1234
Connecting to 127.0.0.1:8080 as tfuchinecco...
Connection established with server
command requires admin privileges
```

Figura 2b: No se permitio agregar un nuevo usuario.

Mala cantidad de parámetros

```
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$ ./bin/client $ $ tfuchinecco 1234 ADD_USER tfuchinecco2
Command 'ADD_USER' expects 2 arguments, got 1
```

Figura 2c: No permite ejecutar sin cantidad correcta de parámetros.

Mala cantidad de longitud de parámetro

```
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$ ./bin/client $ $ tfuchinecco 1234 ADD_USER tfuchinecco2 2222222222222222222222222
222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222
2222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222222
Payload too long
Failed to build command payload
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$
```

Figura 2d: No acepta más de 64 caracteres.

USERS

```
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$ ./bin/client $ $ admin 1234 USERS
Connecting to 127.0.0.1:8080 as admin...
Connection established with server
users count: 2
admin
tfuchinecco
```

Figura 3a: Se listan los usuarios.

Mala cantidad de parámetros

```
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$ ./bin/client $ $ admin 1234 USERS SS
Command 'USERS' expects 0 arguments, got 1
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$
```

Figura 3b: No permite ejecutar sin cantidad correcta de parámetros.

-h

```
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$ ./bin/client -h
Usage: ./bin/client <host> <port> <username> <password> COMMAND [args]
You can use $ to use default values of host and port.
Available commands:
  USERS
  ADD_USER <username> <password>                    (admin only)
  DELETE_USER <username>                            (admin only)
  CHANGE_PASSWORD <username> <new_password>         (admin only)
  STATS
  CHANGE_ROLE <username> <admin|user>               (admin only)
  SET_DEFAULT_AUTH_METHOD <no_auth|username_password> (admin only)
  GET_DEFAULT_AUTH_METHOD                           (admin only)
  VIEW_ACTIVITY_LOG <username>                      (admin only)
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$
```

Figura 4: Se muestran los comandos disponibles.

-v

```
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$ ./bin/client -v
RProxy Client Version 1.0
Developed by GROUP 12
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$
```

Figura 5a: Se muestra la versión y autores.

CHANGE_ROLE

Ejecutado por un administrador

```
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$ ./bin/client $ $ admin 1234 CHANGE_ROLE tfuchinecco admin
Connecting to 127.0.0.1:8080 as admin...
Connection established with server
change_role: role changed successfully
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$
```

Figura 6a: Se modifica correctamente el rol.

Ejecutado por un usuario simple

```
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$ ./bin/client $ $ sdevesa 1234 CHANGE_ROLE tfuchinecco user
Connecting to 127.0.0.1:8080 as sdevesa...
Connection established with server
command requires admin privileges
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$
```

Figura 6b: No se permite modificar el rol.

Mala cantidad de parámetros

```
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$ ./bin/client $ $ admin 1234 CHANGE_ROLE tfuchinecco admin admin
Command 'CHANGE_ROLE' expects 2 arguments, got 3
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$
```

Figura 6c: No permite ejecutar sin cantidad correcta de parámetros.

CHANGE_PASSWORD

Ejecutado por un administrador

```
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$ ./bin/client $ $ admin 1234 CHANGE_PASSWORD tfuchinecco 1212
Connecting to 127.0.0.1:8080 as admin...
Connection established with server
change_password: password changed successfully
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$
```

Figura 7a: Se modifica correctamente.

Ejecutado por un usuario simple

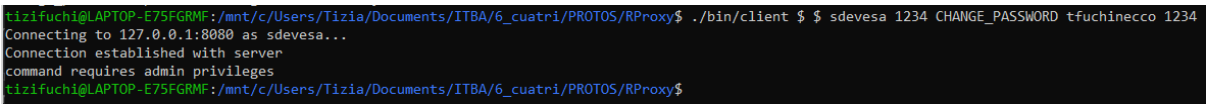

Figura 7b: No se permitio modificar la contraseña.

Mala cantidad de parámetros

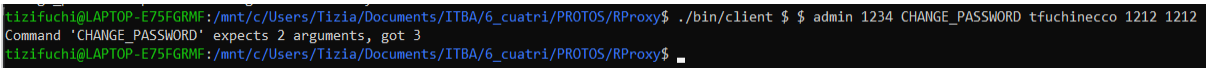

Figura 7c: No permite ejecutar sin cantidad correcta de parámetros

Nueva contraseña con más de 64 caracteres

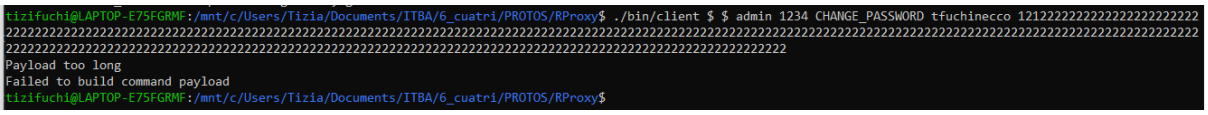

Figura 7d: No modifica la contraseña.

DELETE_USER

Ejecutado por un administrador

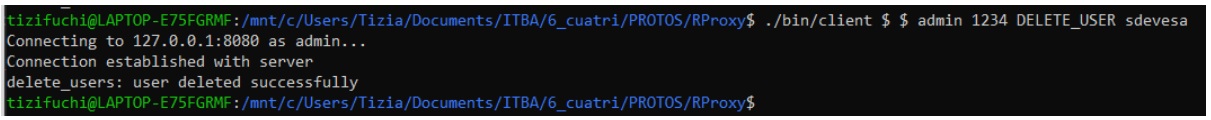

Figura 8a: Se elimina el usuario correctamente.

Ejecutado por un usuario simple

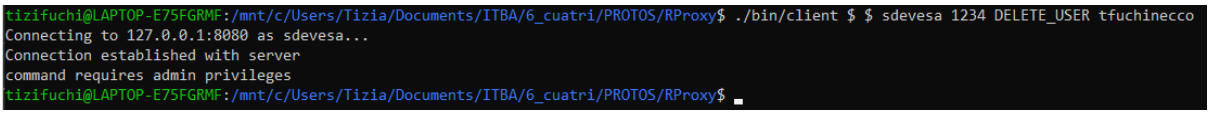

Figura 8b: No permite eliminar el usuario.

Eliminar usuario inexistente

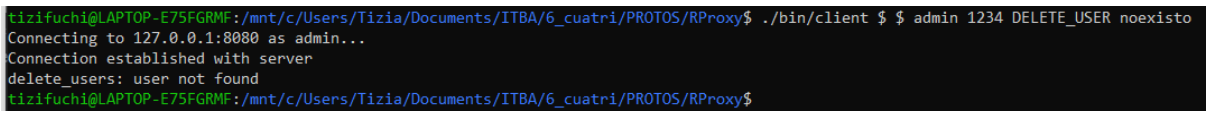

Figura 8c: mensaje de error.

Mala cantidad de parámetros

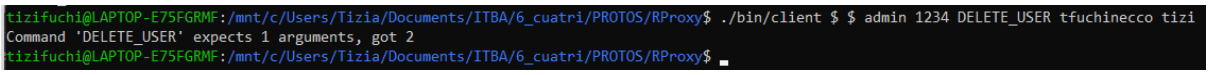

Figura 8d: No permite ejecutar sin cantidad correcta de parámetros.

STATS

```
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$ ./bin/client $ $ admin 1234 STATS
Connecting to 127.0.0.1:8080 as admin...
Connection established with server
Current connections: 0
Total connections: 7
Bytes sent: 1260
Bytes received: 67049

tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$
```

Figura 9a: Permite ejecutar el comando y presenta la información.

Mala cantidad de parámetros

```
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$ ./bin/client $ $ tfuchinecco 1212 STATS parametros
Command 'STATS' expects 0 arguments, got 1
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$
```

Figura 9b: No permite ejecutar sin cantidad correcta de parametros

SET_DEFAULT_AUTH_METHOD

Modificación de método de autenticación a no necesario (por un administrador)

```
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$ ./bin/client $ $ admin 1234 SET_DEFAULT_AUTH_METHOD no_auth
Connecting to 127.0.0.1:8080 as admin...
Connection established with server
set_default_auth_method: default auth method set successfully
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$ ./bin/client $ $ admin 1234 ADD_USER noAuthNeeded 1234
Connecting to 127.0.0.1:8080 as admin...
Connection established with server
add_user: user added successfully
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$
```

Figura 10a: Demostración del correcto funcionamiento permite que un usuario simple ejecute funciones exclusivas para administradores (por ejemplo ADD_USER).

Modificación de metodo de autenticacion a necesario (por un administrador)

```
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$ ./bin/client $ $ admin 1234 SET_DEFAULT_AUTH_METHOD username_password
Connecting to 127.0.0.1:8080 as admin...
Connection established with server
set_default_auth_method: default auth method set successfully
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$ ./bin/client $ $ userSimple 1234 ADD_USER noVaAPoder 1234
Connecting to 127.0.0.1:8080 as userSimple...
Connection established with server
command requires admin privileges
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$
```

Figura 10b: Demostración del correcto funcionamiento NO permite que un usuario simple ejecute funciones de exclusivas de administrador (por ejemplo ADD_USER).

Usuario simple intenta ejecutar

```
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$ ./bin/client $ $ userSimple 1234 SET_DEFAULT_AUTH_METHOD no_auth
Connecting to 127.0.0.1:8080 as userSimple...
Connection established with server
command requires admin privileges
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$
```

Figura 10c: No permite la ejecución a un usuario no administrador.

Mala cantidad de parámetros

```
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$ ./bin/client $ $ admin 1234 SET_DEFAULT_AUTH_METHOD username_password no_auth
Command 'SET_DEFAULT_AUTH_METHOD' expects 1 arguments, got 2
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$
```

Figura 10d: No permite ejecutar sin cantidad correcta de parámetros.

GET_DEFAULT_METHOD

Ejecutado por un administrador

```
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$ ./bin/client $ $ admin 1234 GET_DEFAULT_AUTH_METHOD
Connecting to 127.0.0.1:8080 as admin...
Connection established with server
Default auth method: username_password

tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$
```

Figura 11a: Permite ejecutar y responde el método en uso.

Ejecutado por un usuario simple

```
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$ ./bin/client $ $ tfuchinecco 1212 GET_DEFAULT_AUTH_METHOD
Connecting to 127.0.0.1:8080 as tfuchinecco...
Connection established with server
command requires admin privileges
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$
```

Figura 11b: No permite ejecutar.

Pasando parámetros

```
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$ ./bin/client $ $ admin 1234 GET_DEFAULT_AUTH_METHOD parametro
Command 'GET_DEFAULT_AUTH_METHOD' expects 0 arguments, got 1
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$
```

Figura 11c: No debe recibir parámetros.

VIEW_ACTIVITY_LOG

Ejecutado por un administrador

```
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$ ./bin/client $ $ admin 1234 VIEW_ACTIVITY_LOG tfuchinecco
Connecting to 127.0.0.1:8080 as admin...
Connection established with server
Historial de accesos para tfuchinecco (4 registros):
2025-07-15 00:22:55 - example.com
2025-07-15 00:23:30 - www.directvgo.com
2025-07-15 00:23:56 - www.correoargentino.com.ar
2025-07-15 00:24:45 - www.itba.edu.ar

tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$
```

Figura 12a: Presenta la cantidad de accesos junto con la fecha, hora y el dominio/IP.

Ejecutado por un usaurio simple

```
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$ ./bin/client $ $ tfuchinecco 1212 VIEW_ACTIVITY_LOG tfuchinecco
Connecting to 127.0.0.1:8080 as tfuchinecco...
Connection established with server
command requires admin privileges
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$
```

Figura 12b: No permite ejecutar.

Logs de usuario no existente

```
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$ ./bin/client $ $ admin 1234 VIEW_ACTIVITY_LOG noUser
Connecting to 127.0.0.1:8080 as admin...
Connection established with server
view_activity_log: user not found
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$
```

Figura 12c: Indica que no existe el usuario

Mala cantidad de parámetros

```
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$ ./bin/client $ $ admin 1234 VIEW_ACTIVITY_LOG tfuchinecco parametroExtra
Command 'VIEW_ACTIVITY_LOG' expects 1 arguments, got 2
tizifuchi@LAPTOP-E75FGRMF:/mnt/c/Users/Tizia/Documents/ITBA/6_cuatri/PROTOS/RProxy$
```

Figura 12d: No permite ejecutar sin cantidad correcta de parámetros

Diseño del proyecto.

El diseño del proyecto se basa en una arquitectura cliente-servidor que respeta el protocolo *SOCKS5*, y que implementa las fases de handshake, autenticación y solicitud de conexión. El servidor principal (*socks5v*) actúa como un proxy *TCP* que acepta conexiones entrantes, interpreta solicitudes *"connect"* y redirige el tráfico hacia el destino deseado. Esta implementación fue estructurada en distintos módulos, dividiendo responsabilidades entre el análisis de los mensajes, el manejo de los estados y el procesamiento de cada etapa del protocolo. Además, se desarrolló un cliente, el cual se conecta a través de un socket *TCP* separado del flujo de *SOCKS5*. Este cliente permite a un usuario autenticarse y ejecutar comandos de monitoreo y gestión, como la consulta o modificación de usuarios y la obtención de métricas (si es administrador).

Por otro lado, se diseñó un módulo específico para el manejo de usuarios que administra en memoria las credenciales y permisos, y ofrece funciones para agregar, eliminar o autenticar usuarios. El servidor se basa en un selector de eventos NO bloqueante implementado a través de *select(),* lo que permite manejar múltiples clientes en paralelo.

Entre las decisiones clave de diseño se destaca el uso de un parser por estados para la interpretación de mensajes, implementado mediante punteros a funciones para evitar estructuras switch anidadas y mejorar la claridad/calidad del código. Asimismo, el protocolo de administración se diseñó como una extensión propia, con capacidad para escalar con nuevos comandos. En todo momento se respetaron las restricciones del entorno del trabajo práctico, evitando el uso de librerías externas que realicen el trabajo pedido y manteniendo buffers estáticos, por ejemplo.

Decisiones de implementación

Almacenamiento de usuarios:

Para el almacenamiento de usuarios y contraseñas, se implementó un vector estático en memoria, el cual conserva la información durante toda la ejecución del servidor, hasta el momento de su reinicio. Esta decisión permite un manejo sencillo y eficiente de las credenciales, sin depender de almacenamiento externo o sistemas de persistencia.

Con el objetivo de optimizar el acceso a los datos, se desarrolló una función de hash que permite ubicar rápidamente a un usuario dentro del vector. Esto reduce significativamente el tiempo de búsqueda en comparación con una búsqueda secuencial, lo que resulta especialmente útil cuando se maneja un conjunto moderado de usuarios.

En conclusión, la elección de un vector estático con acceso hash busca rendimiento y simplicidad. Esta solución cumple con los objetivos funcionales del proyecto, y deja abierta la posibilidad de incorporar mecanismos más robustos de persistencia o seguridad en versiones futuras.

Comandos soportados de la fase Requests (RFC 1928):

En el protocolo SOCKS5 (RFC 1928), la fase de request admite tres comandos posibles: CONNECT, BIND y UDP ASSOCIATE. En esta implementación se optó por soportar únicamente el comando CONNECT, que es el más comúnmente utilizado, ya que permite establecer un túnel TCP entre el cliente y un servidor de destino.

Esta decisión se debe principalmente a querer reducir la complejidad. El comando CONNECT cubre el caso de uso más frecuente y esencial en un proxy SOCKS5: el reenvío de conexiones TCP. Al enfocarse exclusivamente en este comando, se simplifica significativamente la lógica del servidor, reduciendo la complejidad del manejo de sockets, y la validación de estados.

Al excluir BIND y UDP ASSOCIATE, la implementación gana en simplicidad, sin perder funcionalidad esencial para los casos de uso más comunes.

Tamaño de los buffers:

Se optó por utilizar buffers de tamaño fijo con el objetivo de evitar alocaciones dinámicas de memoria durante la ejecución del programa. Esta decisión se debe principalmente a criterios de simplicidad, rendimiento y control sobre el uso de recursos.

El uso de buffers fijos permite una implementación más simple y predecible, al eliminar la necesidad de gestionar memoria dinámica mediante llamadas a funciones como malloc, free o realloc. Esto reduce el riesgo de errores relacionados con la gestión de memoria, como memory leaks.

Por otro lado, esta decisión impone ciertas limitaciones en términos de flexibilidad. En particular, si el tamaño del buffer no se ajusta adecuadamente al tamaño real de los datos esperados, puede haber desperdicio de memoria (si es demasiado grande) o riesgo de truncamiento de datos y errores en la comunicación (si es demasiado pequeño). Para mitigar estos riesgos, se eligieron tamaños de buffer suficientemente grandes.

En resumen, se priorizó la eficiencia y robustez de la implementación frente a la adaptabilidad dinámica.