Protocolos de Comunicación
Informe TPE

Grupo 6

Integrantes:

Méndez, Ignacio Alberto	-	59058
Griggio, Juan Gabriel	-	59092
Villanueva, Ignacio	-	59000
Zuberbuhler, Ximena	-	57287

Descripción de la aplicación

La aplicación desarrollada en este trabajo práctico es un Proxy SOCKSv5[1], capaz de atender a múltiples conexiones concurrentes con operaciones no bloqueantes. Esto hace que ningún cliente que se conecte al proxy tenga que ver degradada su experiencia debido a que el proxy se bloqueó atendiendo a otro cliente.

Autenticación SOCKSv5

Para la autenticación de para el uso del servicio proxy, se siguieron los lineamientos del RFC de Socksv5.De los posibles métodos para autenticarse en SOCKSv5, nuestra aplicación soporta:

- Sin autenticación.
- Autenticación con usuario y contraseña.[2]

Requests SOCKSv5

De los posibles comandos para hacer requests en SOCKSv5, nuestra aplicación soporta:

- Comando “CONNECT”.

De los posibles “Address Types” que se pueden indicar en un request SOCKSv5, nuestra app soporta:

- Dirección IPv4.
- Dirección IPv6.
- FQDN.

Flujo de la aplicación

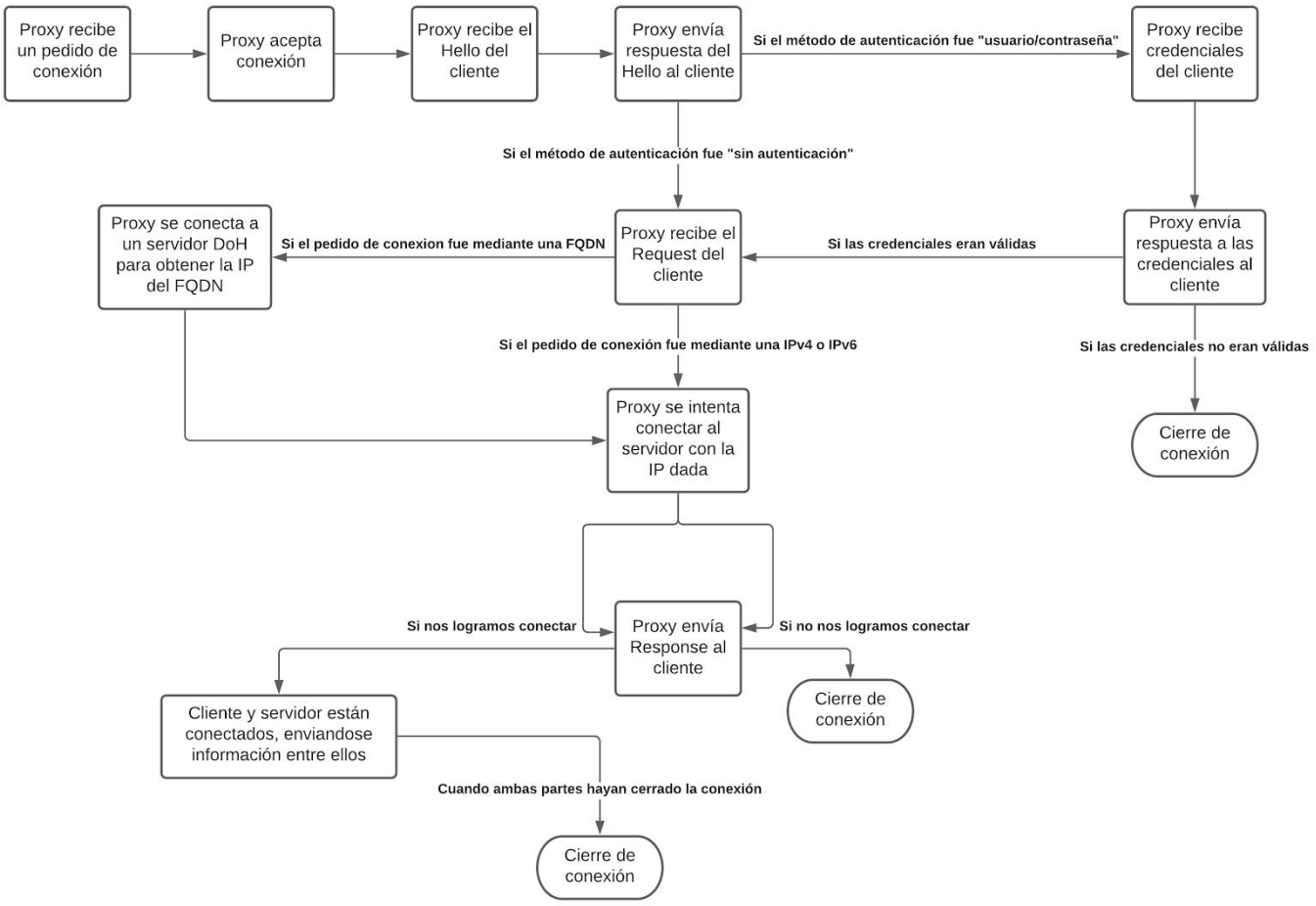

Figura 1: diagrama de flujo de la aplicación

[1] Protocolo SOCKSv5: https://tools.ietf.org/html/rfc1928
[2] Autenticación usuario/contraseña para el protocolo SOCKSv5: https://tools.ietf.org/html/rfc1929

Cabe aclarar que en cada estado en el que el proxy recibe algo del cliente y lo procesa, si lo recibido es inválido en cuanto al protocolo SOCKSv5 se refiere, el proxy le responderá con un mensaje apropiado de error. Esto puede suceder, por ejemplo, si el cliente envía un Request con un comando no soportado por la aplicación.

Registro de accesos

Cuando una conexión cliente-servidor es establecida, el proxy lo informa imprimiendo en pantalla de manera no bloqueante el registro de acceso. Un posible ejemplo es el siguiente:
2020-11-15T10:06:07Z nacho A 127.0.0.1 53742 foo.leak.com.ar 80 1
En donde se informa:

1. El momento de establecimiento de conexión.
2. Las credenciales de autenticación del cliente.
3. Tipo de registro (siempre el caracter “A”).
4. IP del cliente (puede ser IPv4 o IPv6)
5. Puerto del cliente.
6. IPv4/IPv6/FQDN de a dónde nos conectamos.
7. Puerto destino.
8. Status de la conexión (son los posibles “replies” que el protocolo SOCKSv5 envía en su Response hacia el cliente).

Registro de credenciales

Una vez que la conexión cliente-servidor es establecida, ambas partes pueden empezar a enviar datos entre sí. Esos datos pueden estar encapsulados en algún protocolo de aplicación, como lo es HTTP. Nuestro proxy se encarga de analizar los datos que él mismo forwardea, almacenando las credenciales que ambas partes usan para autenticarse, cuando se usan los protocolos HTTP y POP3.
Esto quiere decir que, si una de las partes le quiso enviar a la otra un paquete HTTP ó POP3 con información de autenticación, nuestro proxy almacenará esas credenciales y las imprimirá en pantalla de manera no bloqueante.
Un posible ejemplo:
2020-11-15T10:06:07Z nacho P HTTP foo.leak.com.ar 80 juan pass123
En donde se informa:

1. Momento en el que se logró obtener las credenciales.
2. Nombre del usuario que hace el requerimiento.
3. Tipo de registro (siempre el caracter “P”).
4. Tipo de protocolo (HTTP o POP3).
5. IPv4/IPv6/FQDN de a dónde nos conectamos.
6. Puerto destino.
7. Usuario descubierto.
8. Contraseña descubierta

DoH

Para resolver FQDN se implementó DNS over HTTP. En principio esto requiere de la implementación de parsers para ambos HTTP y DNS, los cuales requirieron de mecanismos para parsear sin la necesidad de tener el mensaje entero, debido a que los paquetes recibidos pueden estar fraccionados. El flujo para resolver el FQDN comienza por arrancar la conexión con el servidor de DoH especificado, donde se crea el paquete de query el cual se debe enviar. El paquete creado es un request HTTP de tipo POST, el cual contiene como cuerpo una única pregunta de DNS. Una vez enviado el paquete por completo, se espera

hasta recibir una respuesta de parte del servidor, el cual primero se parsea la respuesta HTTP, y luego el DNS. En caso de éxito, se prosigue al intentar conectar al servidor destino, probando todas las respuestas del DNS, hasta obtener una conexión o agotar las respuestas. En caso de recibir un reply code de "*Domain name does not exist."* se prosigue a responder al cliente con un código de *Network Unreachable.*

En la actual implementación nuestra, solo se puede hacer queries de una única respuesta, lo cual consideramos suficientemente eficiente para la mayoría de los casos, ya que primero se pregunta por los IPv4 adheridos al FDQN, y en la minoria de los casos donde solo se pueda resolver por IPv6, se prosigue a crear un query para IPv6 una vez todas las direcciones de IPv4 respondidas por el servidor hayan fallado la conexión. Finalmente, en cada request que se le haga al servidor DoH, se abre y cierra un socket nuevo.

Descripción de myProtocol

Nuestro protocolo fue descripto en el RFC *Proxy SOCKSv5 configuration protocol*. Es un protocolo de aplicación orientado a conexión sobre SCTP. Este es utilizado para obtener métricas como cantidad de conexiones históricas, cantidad de conexiones concurrentes, cantidad de bytes transmitidos y para hacer cambios de configuración como el tamaño del buffer. Este protocolo requiere autenticación mediante usuario y contraseña.

Se implementó un server myProtocol que atiende las conexiones de nuestro protocolo dentro de nuestra aplicación, verificando que el usuario que se quiere conectar sea el administrador indicado cuando se comienza a ejecutar el programa (mediante el argumento -U). Luego de la autenticación, se procesa el request y se envía una respuesta. Este server atiende conexiones por defecto en el puerto 8080 (y puede ser modificado mediante el argumento -P). En el caso del cambio de tamaño de buffer, si el nuevo tamaño de buffer es menor a un mínimo deja el tamaño anterior. El mínimo tamaño de buffer decidido es 300, ya que con tamaños menores el funcionamiento del proxy puede ser indefinido.

También se implementó un cliente de myProtocol, el cual se ejecuta por línea de comando con los siguientes argumentos: puerto del server, hostname del server, nombre de usuario, contraseña y la opción de métrica o configuración y sus parámetros (de ser necesarios). Este cliente se conecta al server, realiza la autenticación, envía el request y luego devuelve en pantalla la respuesta que recibe por parte del server.

Problemas encontrados durante el desarrollo

A continuación enumeramos algunos de los mayores problemas encontrados durante el desarrollo:

- Debido al tamaño del proyecto, nos encontramos de a momentos dificultoso el manejo de memoria de forma eficiente. En sí esto se podría haber mejorado con un mayor periodo de planificación y posiblemente con el uso de TDAs para generar mayor abstracción.
- Debido a las limitaciones de tiempo, se tuvo que ceder cierta eficiencia al tiempo de procesar datos, creemos que con mayor tiempo se podría mejorar aún más la eficiencia del mismo.
- Al intentar conectarnos al servidor origen, no entendíamos al principio por qué siempre nos devolvía error. Nos dimos cuenta al poco tiempo que se debía a que todos nuestros sockets están marcados como “no bloqueantes”. Por lo tanto, la acción de “connect()” era bloqueante y nos devolvía el error EINPROGRESS. La solución fue atrapar ese error y suscribir a nuestro socket del servidor origen para escritura para luego revisar si la conexión fue exitosa o no.

Limitaciones de la aplicación

A continuación enumeramos ciertas limitaciones de la aplicación:

- El parseo de contraseñas de los protocolos POP3 y HTTP en casos extremos pueden llegar a registrarse de forma errónea. Un ejemplo, en POP3, si el cliente envía usuario y contraseña, previo a que el servidor POP3 se identifique, no se registraron los datos.
- En DoH, se utilizan buffers de tamaño estático para recibir los paquetes. En el caso límite donde se reciba un paquete extremadamente grande, fallara la consulta. El tamaño del mismo puede ser modificado mediante las configuraciones.
- Servidor SOCKS tiene un máximo arbitrario de 501 conexiones que puede ser incrementado. En la práctica este máximo puede variar por distintos aspectos, tales como velocidad con la cual se conectan los usuarios o si se usa o no doh. Más información sobre esto en la sección de pruebas de stress.

Pruebas de Stress

Se llevaron a cabo varias pruebas de stress, para encontrar cuales son las limitaciones prácticas del servidor SOCKS. A continuación enumeramos cuáles fueron las pruebas y cuales son nuestros resultados:

- Prueba de velocidad de conexiones consecutivas. En esta prueba se buscó encontrar qué tan rápido el servidor es capaz de aceptar nuevas conexiones. Para esto se tomaron 2 variables en consideración, intervalo entre cada nueva conexión, y cantidad de conexiones. Utilizando un servidor HTTP local, se logró conseguir 5000 conexiones (tomando en consideración que no todas son concurrentes), en donde el intervalo entre nuevas conexiones es practicamente instantaneo (limitado por qué tan rápido la aplicación de prueba de stress es capaz de crear nuevas conexiones), y el servidor proxy fue capaz de manejar todas las conexiones, sin ningún tipo de pérdida.
- Prueba de tiempo de descarga. En esta prueba se utilizó un paquete de prueba de 20MB, 200MB y 1GB para medir el tiempo de descarga utilizando el proxy y sin el proxy. Los resultados son los siguientes

Tamaño	Con Proxy	Sin Proxy	Porcentaje de Cambio
1GB	2.905s	0.827s	351%
200MB	0.609s	0.172s	354%
20MB	0.108s	0.022s	490%

Se puede ver que hay un incremento considerable del 400% en promedio. En si es de esperarse un incremento, debido a que se duplican las lecturas y escrituras, además del procesamiento interno del proxy.

- Conexiones concurrentes. Para esta prueba se utilizó Jmeter, donde se hace un llamado a un servidor local de nginx el cual contiene un paquete de 20 Mb para descargar. Debido, a lo que suponemos, de la limitación del servidor web, no se lograron establecer las teóricas 500 conexiones de máximo límite. Se logró conseguir entre 200 conexiones de forma estable, donde todo el archivo es transferido en todas la conexiones. En casos extremos se ha logrado llegar entre 300 y 400 conexiones, donde se ha encontrado transferencias parciales o el servidor web no responde. Poniendo de lado el archivo transferido, se puede notar que las 500 conexiones son registradas por el servidor proxy y el mismo género los 500 requests al servidor web.

Posibles extensiones

A continuación enumeramos posibles extensiones las cuales se podrían hacer a la aplicación en el futuro:

- Pool de conexiones persistentes de DoH. Esto permitiría evitar tener que iniciar y finalizar conexiones innecesariamente con el servidor DoH, mejorando la velocidad de la aplicación.

Conclusiones

En conclusión, se logró crear un servicio proxy SOCKSv5 junto a un servicio mediante SCTP para configurar y monitorear el mencionado, los cuales son capaces de manejar múltiples conexiones de forma no bloqueante. Hay varios aspectos de los cuales estos pueden ser mejorados, pero consideramos que en el marco de tiempo establecido, se lograron los resultados esperados por la cátedra.

Ejemplos de prueba

Todos los siguientes ejemplos se deben correr usando el proxy con sus valores default. Es decir, sin pasarle ningún parámetro extra por línea de comandos.

curl -x socks5h://127.0.0.1:1080 foo.leak.com.ar

- Va a devolver el contenido de esa página web, y se va a ver cómo el proxy loguea que la conexión fue exitosa.

curl -x socks5h://127.0.0.1:1080 google.com

- ídem caso anterior.

nc -x 127.0.0.1 127.0.0.1 110 (Requiere de un servidor POP3 en localhost)

- Nos vamos a lograr conectar al servidor POP3. El proxy loguea que la conexión fue exitosa.

nc -x127.0.0.1 127.0.0.1 9090 (con un netcat levantado en 127.0.0.1:9090 en otra terminal)

- Nos vamos a lograr conectar a la otra terminal. El proxy loguea que la conexión fue exitosa.

nc -x127.0.0.1 ::1 9090 (con un netcat levantado en ::1:9090 en otra terminal)

- Ídem caso anterior.

nc -x[::] 127.0.0.1 9090 (con un netcat levantado en 127.0.0.1:9090 en otra terminal)

- Ídem caso anterior.

nc -x[::] ::1 9090 (con un netcat levantado en ::1:9090 en otra terminal)

- Ídem caso anterior.

Guía de instalación

Si se requiere correr el proyecto en su computadora, hay que seguir una serie de pasos:

1. Clonar el repositorio de Git a su computadora.
2. Se utilizó CMAKE para compilar el proyecto, por lo que se deben ejecutar los siguientes comandos para compilar:

```
mkdir build
cd build
cmake ../
make
```

3. Los archivos ejecutables se encontraran en build/bin/, por lo cual con el siguiente comando se podrá inicializar el servidor

```
./bin/socks5d [ARGS]
```

Por otro lado, para correr el cliente hay que posicionarse en la carpeta /build y ejecutar el siguiente comando:

```
./bin/client <port_number> <hostname> <username> <password> <option> [parametro]
```

Instrucciones para la configuración

Para configurar el servidor Socks5 y el servidor myProtocol, en su mayoría se pueden utilizar los argumentos de línea de comando establecidos en el man page creado por la cátedra. En caso de no encontrarse en el mismo la configuración necesaria, en la dirección *./src* se puede encontrar un archivo header llamado *config.h* el cual establece todas las configuraciones por defecto del servicio.

Ejemplos de configuración y monitoreo

Para estos ejemplos, hay que correr el proxy con la opción “-U” para añadirle un usuario de administrador. Por ejemplo, se puede correr así: *./socks5d -Uadmin:admin* y luego en los siguientes ejemplos reemplazar el “username” y “password” por “admin”.

./client 8080 localhost username password 1

- Va a devolver la cantidad de conexiones desde el inicio del servidor, en formato ASCII.

./client 8080 localhost username password 2

- Va a devolver la cantidad de conexiones concurrentes del servidor en el momento de ejecución del comando, en formato ASCII.

./client 8080 localhost username password 3

- Va a devolver la cantidad de bytes transferidos desde el inicio del servidor, en formato ASCII.

./client 8080 localhost username password 4 2500

- Se modifica en el servidor el tamaño del buffer a 2500 bytes y devuelve el tamaño ingresado si la operación fue exitosa, en formato ASCII.

./client 8080 localhost username password 4 250

- El tamaño del buffer en el servidor queda igual y devuelve un código de error con un mensaje “Buffer size too small”

Créditos de Código Fuente

Librería Base64: https://nachtimwald.com/2017/11/18/base64-encode-and-decode-in-c/
Librería Buffer: Librería de la cátedra
Librería Args: Librería de la cátedra
Librería Netutils: Librería de la cátedra
Librería Selector: Librería de la cátedra
Librería STM: Librería de la cátedra