ITBA
PROTOCOLOS DE COMUNICACIÓN
TPE: SOCKSv5

Grupo 2
Santiago Dylan Terenziani - 57240
Enrique Tawara - 58717
Sofía Picasso - 57700

Índice

DESCRIPCIÓN DETALLADA DE LOS PROTOCOLOS Y APLICACIONES DESARROLLADAS

PROXY

El proxy se desarrolló siguiendo los lineamientos establecidos por los RFC 1928 y 1929. El primero especifica el intercambio de paquetes que ocurre cuando un cliente intenta conectarse con un proxy Socksv5.

Inicialmente, el cliente debe enviar al puerto donde el proxy espera las conexiones entrantes (por defecto, el puerto 1080) un mensaje con la siguiente estructura:

VER	NMETHODS	METHODS
1	1	1 to 255

La segunda fila indica el tamaño en bytes de cada campo. El campo VER debe valer 5 (Pues representa la versión del protocolo), NMETHODS contiene la cantidad de “method identifier octets” que aparecen en METHODS, y finalmente, METHODS contiene los identificadores de métodos que el usuario puede utilizar.

El server selecciona de uno de los métodos en METHODS y envía un mensaje de selección de método:

VER	METHOD
1	1

Si el método es 0xFF, ninguno de los métodos listados por el cliente son aceptables, y el cliente debe cerrar la conexión.
Los valores que puede tomar METHOD son:

0x00 NO AUTHENTICATION REQUIRED
0x01 GSSAPI
0x02 USERNAME/PASSWORD
0x03 to 0x7F IANA ASSIGNED
0x80 to 0xFE RESERVED FOR PRIVATE METHODS
0xFF NO ACCEPTABLE METHODS

En caso de que el método valga 0x02, se pasará a una etapa de autenticación donde el usuario se identificará y será aceptado o rechazado por el servidor. Según el RFC 1929, la autenticación opera de la siguiente forma, comenzando por un mensaje del cliente:

VER	ULEN	UNAME	PLEN	PASSWD
1	1	1 to 255	1	1 to 255

Donde VER es la versión de la sub-negociación (0x01), ULEN es la longitud del campo UNAME, UNAME es el nombre de usuario en el OS source, PLEN es la longitud del campo PASSWD, y PASSWD es la contraseña.

El server verificará el usuario y contraseña provistos y enviará el siguiente response, donde VER contendrá el valor 0x05 como se mencionó anteriormente, y STATUS valdrá 0x00 en caso de éxito. Cualquier otro valor de STATUS representa un error y por lo tanto la conexión debe cerrarse.

VER	STATUS
1	1

Una vez autenticado (o en el caso de que el servidor no requiera autenticación, una vez que se le respondió al cliente tras su primer mensaje) el cliente enviará los detalles de su request de la siguiente forma:

VER	CMD	RSV	ATYP	DST.ADDR	DST.PORT
1	1	0x00	1	Variable	2

Donde VER es la versión del protocolo (de nuevo, 5), CMD es el comando a ejecutar, RSV es un espacio reservado, ATYP es el tipo de dirección (IPv4, IPv6 o un nombre de dominio a resolver), DST.ADDR es la dirección de destino y DST.PORT es el puerto de destino (en orden network octet).

Los valores posibles para CMD son 0x01 (Connect), 0x02 (Bind) y 0x03 (UDP Associate).
Los valores posibles para ATYP son 0x01 (IPv4), 0x03 (Domain name) y 0x04 (IPv6). En el caso de que ATYP sea un nombre de dominio, el primer byte de DST.ADDR contendrá la longitud del nombre, que comienza en el byte siguiente.

El server evaluará el request y devolverá un paquete con este formato:

VER	CMD	RSV	ATYP	BND.ADDR	BND.PORT
1	1	0x00	1	Variable	2

Donde VER es la versión (una vez más, 5), CMD es 0x00 en caso de éxito (o un código de error en caso de falla), RSV es un byte reservado, ATYP es el tipo de dirección, y BND.ADDR y BND.PORT representan la dirección y puerto del servidor a donde fue conectado el cliente.
Al igual que en Pampero, estos campos serán nulos al conectarse a nuestro servidor, ya que al no soportar los comandos BIND y UDP ASSOCIATE, la información contenida en esos campos es irrelevante.

Los valores que puede tomar CMD en este caso son:

0x00	SUCCESS
0x01	General SOCKS server failure
0x02	Connection not allowed by ruleset
0x03	Network unreachable
0x04	Host unreachable
0x05	Connection refused
0x06	TTL expired
0x07	Command not supported
0x08	Address type not supported
0x09 to 0xFF	Unassigned

Si el mensaje enviado al usuario fue una aprobación, se pasa a la etapa de copiado, donde el cliente enviará al proxy los paquetes que normalmente le enviaría al servidor donde quería conectarse. El deber del proxy es pasar los paquetes de un lado a otro a medida que lleguen, opcionalmente analizando los datos enviados como se hizo en este caso para obtener usuarios y contraseñas enviados a través del proxy.

La implementación del protocolo consiste en crear un socket pasivo que espera conexiones entrantes. Cuando recibe una conexión, crea una estructura llamada socks5 donde se almacenarán datos relevantes a la conexión que se está creando.

Esto incluye las direcciones y puertos de los hosts conectados (y los file descriptors asociados con cada uno), el usuario que realizó la conexión, la máquina de estados que describe el funcionamiento del proxy y su estado actual, información utilizada por los parsers de los paquetes mencionados anteriormente, etc.

Al crear esta estructura, la misma se inicializa en el estado HELLO_READ, el cual abarca el saludo inicial del cliente que pide conectarse. En este estado, se leerán los bytes que envíe el cliente y se procesará una respuesta acorde a la información enviada. Una vez generada la respuesta, se pasará al próximo estado.

El segundo estado es HELLO_WRITE. Una vez que el cliente esté libre para recibir mensajes, se le enviará la respuesta armada para pasar al siguiente paso.

Si el usuario debe autenticarse, se pasará al estado AUTH_READ, donde se procesarán los datos ingresados por el cliente y se generará una respuesta acorde, habilitando al usuario si las credenciales son válidas, y rechazándolo en caso de no ser un usuario registrado.

Similarmente al intercambio anterior, el estado AUTH_WRITE se encarga de responderle al usuario con esta respuesta cuando el mismo esté listo para recibir una respuesta del proxy. Si el usuario es rechazado, la estructura creada previamente es liberada y la conexión se cierra.

Una vez que el usuario es bienvenido a utilizar el proxy, pasa al estado REQUEST_READ, donde se procesa el pedido de conexión del usuario. En caso de recibir una IP, el proxy intentará conectarse a la misma y en caso de lograrlo, pasará al estado REQUEST_WRITE. En cambio, si el paquete contenía un dominio a resolver, el estado será REQUEST_RESOLVE.

El mismo consiste en la creación de un hilo donde se resolverá el nombre a alguna IP. Una vez que esa resolución esté lista, se alertará al selector que finalizó una tarea bloqueante, y se ejecutará el handler correspondiente donde se revisará si la operación ocurrió con éxito o no. Si la resolución fue exitosa, se pasa al estado REQUEST_WRITE, donde, al igual que en los Write anteriores, se envía una respuesta acorde al cliente.

Si la conexión fue establecida con éxito, se pasará al estado COPY, donde toda información que llegue al proxy de cualquier lado (cliente o servidor de origen) será enviada al otro, no sin antes parsear los datos enviados en búsqueda de contraseñas (siempre y cuando esta opción esté activada).

Cuando alguno de los dos sockets cierre la conexión, se pasará a un estado terminal, donde se guardará en un pool, de ser posible, la estructura socks5 creada para ser reutilizada por otra conexión futura. En caso de que el pool supere el tamaño máximo, la estructura será liberada inmediatamente, sin esperar a que finalice la ejecución del server.

Cuando el servidor detecte una señal SIGTERM O SIGINT, cerrará las conexiones abiertas y liberará la memoria utilizada.

DoH

Comúnmente la resolución de nombres se realiza mediante un request a un DNS server. Este método presenta desventajas a nivel de privacidad y seguridad ante situaciones como DNS spoofing.

Según el RFC 8484, el protocolo DoH (DNS over HTTPS) consiste en encapsular un mensaje DNS en un mensaje HTTPS, siendo una alternativa más segura y que mejor resguarda la privacidad del cliente al aprovechar los beneficios que brinda HTTPS.

El mensaje HTTP puede ser un método GET o un método POST. En el caso del GET, se codifica la query DNS en Base64url, mientras que POST pone el query en el cuerpo del objeto HTTP. En este caso se debería especificar Content-Type como application/dns-message.

Ambos métodos tienen sus ventajas y desventajas pero se decidió utilizar el método POST para evitar la codificación del query.

Es necesario requerir que el mensaje de respuesta sea de tipo application/dns-message y por ende se agrega el header “Accept: application/dns-message”

El content-type application/dns-message es un mensaje DNS en el formato “full wire” definido en el RFC 1035. Dicho formato consiste en:

Header
Question (pregunta al name server)
Answer (rr de respuesta)
Authority (rr a la autoridad)
Additional (rr con información adicional)

Tanto en query como en response el formato del mensaje es igual. Los campos de mayor interés son los 3 primeros.

El header guarda 12 bytes de información acerca del mensaje donde:

ID: 16 bits, en DoH se recomienda que sea 0

QR: 1 bit, 0 para query, 1 para response

OPCODE: 4 bit, 0 para standard query

AA: 1 bit, Authorative Answer - para respuestas

TC: 1 bit, 0 para indicar que el mensaje no es truncado, 1 para truncado

RD: 1 bit, 1 para desear recursión, ya que queremos que en caso de que retorne un name server, lo consulte de vuelta.

RA: 1 bit, en response define si la recursión es válida en dicho servidor

Z: 3 bits, debe ser 0

RCODE: 4 bits, response code - indica el status del response. 0 indica que no hubo ningún error

QDCOUNT: 16 bits, cantidad de preguntas en dicho mensaje, generalmente 1

ANCOUNT: 16 bits, cantidad de answers en el mensaje

NSCOUNT: 16 bits, cantidad de name server resource records en authority records

ARCOUNT: 16 bits, cantidad de resource records en additional records

El question section está dividido en 3 partes:

- QNAME: longitud variable, nombre del dominio separado en labels. Cada '.' separa un label del otro en el nombre del dominio. Primero se indica la cantidad de octetos que ocupa un label seguido por dicha cantidad de octetos. La longitud de cada label debe ser menor a 64. Se indica con un 0x00 que terminó. No hay padding al final.
- QTYPE: 16 bits, indica el tipo del query, es decir lo que se desea. Se destaca:
  - 0x01 indica A
  - 0x1c indica AAAA
  - 0x05 indica CNAME
  - 0x06 indica SOA
  - 0xff indica ANY
- QCLASS: 16 bits, especifica la clase del query. Para este proyecto se desea la clase IN, representado con 0x01

Las secciones a continuar están presentes en formato RR, que consiste en:

- NAME: Nombre del dominio en cuestión. Puede ser del mismo formato que QNAME (longitud variable) o un puntero (16 bits). Se puede distinguir leyendo los primeros 2 bits, 00 para nombre, 11 para puntero.
- TYPE: 16 bits, tipo de la respuesta (0x01 para A, 0x1c para AAAA, etc.)

- CLASS: 16 bits, 0x01 (IN)
- TTL: 32 bits, tiempo de vida en segundos.
- RDLENGTH: 16 bits, cantidad de octetos de la respuesta.
    - para A vale 16
    - para AAAA vale 40
- RDATA: la respuesta, su longitud varía del tipo de dato indicado por RDLENGTH.
    - en ambos A y AAAA es la dirección IP en sus respectivos formatos

Para la implementación de una función que resuelve nombres mediante DoH, llamada solveDomain, se decidió utilizar un formato idéntico a getaddrinfo a la hora de resolver nombres dado que presentan funcionalidades intercambiables con el fin de facilitar la comprensión de la función. A pesar de ello se agregó el parámetro doh para poder especificar cual es el servidor DoH a consultar, en caso de sea NULL utilizará el servidor DoH por defecto.

Para el armado de requests se utilizó un buffer al cual se le alimenta del request pedido ya que dicha implementación permite un fácil manejo del contenido a enviar y manejar desde donde reanudar en caso de que un solo send no haya podido enviar el mensaje completo.

Para el parseo del response se utilizó otro buffer, dado que el parseo permite analizar un octeto a la vez y aun así poder hacer read de mayor cantidad de datos. También se aprovechó la utilidad adicional para no escribir en una dirección a la cual no pertenece al buffer.

En el caso de que el response DoH no presente direcciones AF_INET o AF_INET6 se decidió llamar a getaddrinfo como plan auxiliar.

CLIENTE

El protocolo SCTP definido para el cliente consiste en un intercambio de cuatro mensajes como máximo. Dos por parte del cliente y dos provenientes del servidor.

Tras conectarse al servidor, el cliente debe enviar un paquete estableciendo la versión del protocolo a utilizar (siendo esta la única versión actual, su valor será 1) y el usuario y contraseña con los cuales autenticarse con el servidor. Su estructura es la misma que la del paquete de autenticación definido en el RFC 1929.

VER	ULEN	UNAME	PLEN	PASSWD
1	1	1 to 255	1	1 to 255

El servidor luego retornará una respuesta repitiendo la versión y retornando un código de error, o 0x00 en caso de que el usuario haya sido autenticado con éxito.

VER	STATUS
1	1

Si la autenticación fue válida, el usuario puede ahora establecer el comando a utilizar mediante el siguiente paquete, donde DATA representa un conjunto de bytes cuya interpretación depende del comando en el campo CMD.

VER	CMD	DATA
1	1	Variable

El comando 0x00 es “Agregar usuario”. Dentro del campo DATA se espera una representación del usuario y contraseña de la siguiente forma:

VER	CMD	ULEN	UNAME	PLEN	PASSWD
1	0x00	1	1 to 255	1	1 to 255

El comando 0x01 es “Listar usuarios”. Ya que el usuario no debe enviar ninguna información adicional, el campo DATA queda vacío.

VER	CMD
1	0x01

El comando 0x02 es “Devolver métricas”. De la misma forma, no hace falta información adicional, por lo que el campo DATA debe ser vacío.

VER	CMD
1	0x02

El comando 0x03 cambia la cantidad máxima de usuarios aceptados por el server. El campo DATA, por lo tanto, contendrá la cantidad de bytes que a leer seguido por el nuevo máximo a establecer. En este caso, la variable de la cantidad máxima de usuarios es un unsigned int que ocupa 4 bytes.

VER	CMD	SLEN	SIZE
1	0x03	1	4

Una vez recibida la instrucción, el servidor intentará procesar la solicitud y generará una respuesta acorde con el siguiente formato, para cada comando mencionado anteriormente.

Si el comando fue añadir un usuario (0x00), se retornará un 0x00 en caso de haberlo agregado con éxito, y un código de error en caso de falla.

VER	CMD	STATUS
1	0x00	1

Para el comando de métricas (0x02), se retornará, después de una repetición de la versión y el comando, tres variables con su tamaño en bytes definido dentro de los campos HLEN, ALEN y BLEN. Esto permite determinar el tipo de datos necesario para almacenar cada valor, siempre siendo valores positivos o nulos.

HIST representa la cantidad de conexiones que se realizaron al servidor con éxito. ACTIVE, por su parte, representa las conexiones activas. Es decir, usuarios que se encuentran conectados al servidor en ese momento. El campo BYTES representa todos los bytes que transfirió el proxy desde un cliente al servidor de origen y viceversa, para todas las conexiones realizadas desde que se inicializó el servidor.

VER	CMD	HLEN	HIST	ALEN	ACTIVE	BLEN	BYTES
1	0x01	1	1 to 255	1	1 to 255	1	1 to 255

Para el comando de listar usuarios (0x01), se retorna LLEN para indicar la longitud de la lista, y dentro de DATA se retornarán la longitud de cada nombre de usuario y su nombre. Para extraer los valores, es necesario leer LLEN para saber de antemano cuántos usuarios se esperan a continuación.

VER	CMD	LLEN	DATA
1	0x02	1	Variable

Por cada usuario que se retorna, la información se representa de la siguiente manera:

ULEN	UNAME
1	1 to 255

Para el comando de modificación de variable (0x03) se retorna el estado de la operación. 0x00 en caso de éxito, cualquier otro valor en caso de error.

VER	CMD	STATUS
1	0x03	1

PROBLEMAS ENCONTRADOS DURANTE EL DISEÑO Y LA IMPLEMENTACIÓN

En las etapas iniciales, cuando se estaba implementando la conexión inicial entre el cliente y el servidor proxy, hubo momentos donde el servidor fallaba o no respondía los mensajes del usuario. Con ayuda de Wireshark, logramos comprender dónde fallaba la comunicación entre ambas partes y quién era responsable del error. Por ejemplo, había casos donde el servidor no generaba una respuesta y dejaba al usuario en espera de la misma. O casos donde la respuesta que generaba no era la esperada, representando un error en el parseo del mensaje del cliente o en la creación de una respuesta acorde.

Un desafío importante durante el desarrollo fue la prueba de conexión. Si bien en las etapas iniciales se intentó tímidamente realizar requests mediante comandos como curl o netcat, el hacerlos uno por uno demostraba ser una prueba demasiado sencilla para lo que debería ser capaz de soportar el servidor.

Para poner a prueba el servidor, se habilitó temporalmente la falta de autenticación de forma tal que los navegadores que soporten el uso de servidores socks5, pero no la autenticación contra los mismos, puedan navegar a través del nuestro. De esta forma pudimos verificar que la velocidad no se veía afectada severamente al pasar todos los paquetes por el proxy, y que se podían realizar múltiples requests simultáneamente, con la información siendo transferida de manera no bloqueante.

En la versión final, se forzó al usuario a autenticarse, por lo que las pruebas debieron ser realizadas mediante un script que realiza varios requests simultáneos para verificar que el servidor pueda atender una cantidad elevada de clientes autenticados al mismo tiempo.

Otro conflicto que se encontró durante el desarrollo fue la creación de dos sockets pasivos en la inicialización del servidor. Por defecto, el mismo escucha en todas las interfaces, tanto IPv4 como IPv6. Esto hace que al bindear un socket cuya dirección es “::” (la cual representaba todas las interfaces para IPv6) también la bindeara a “0.0.0.0” (equivalente para IPv4).

Esto hacía imposible bindear dos sockets distintos para escuchar a todas las interfaces de distinto tipo de IP, ya que en ejecución ocurría un error al bindear dos veces a la misma dirección 0.0.0.0.

La solución fue añadir una configuración al socket IPv6 tal que solo acepte conexiones de ese tipo, dejando así la dirección 0.0.0.0 al socket IPv4 únicamente.

Originalmente, el registro de usuarios iba a ser persistente, pero dada la complejidad de manejar archivos de manera no-bloqueante, y de parsear una lista larga en busca de un par usuario-contraseña, se optó por tener un registro de tamaño limitado en memoria. Dar marcha atrás con la decisión no fue ideal, pero se consideró que priorizar las demás funcionalidades era más importante.

La disección de contraseñas, si bien no fue una tarea demasiado complicada, resultó más compleja de lo esperado, ya que no siempre se podía esperar un salto de línea de la misma forma. Analizando los paquetes enviados en medio de una conexión POP3, notamos como al usar curl para acceder al servidor de emails, los saltos de línea se representaban con un \r\n, mientras que al utilizar netcat e ingresar los comandos manualmente, los saltos se representaban únicamente con \n. Al descubrir esta diferencia, se debió modificar la máquinas de estados de los parsers de HTTP y POP3 para tener en cuenta ambas posibilidades.

Finalmente, debido a un problema donde no siempre se pueden resolver direcciones IPv6 con un connect, se decidió implementar la solución de dominios de tal forma que en lo posible se tenga disponible una dirección ipv4 dentro de la lista de direcciones ip que devuelve el DoH request.

El rfc 8484 establece que el protocolo DoH consiste en resolver requests dns a través de un request https. Dado que el conocimiento para enviar un request https excede la materia, surgió la necesidad de implementar un método para poder resolver requests dns a través de http.

Para ello, se utilizó un servidor http en nginx que al recibir un request dns, lo envía redirecciona como request https.

Con el fin de no depender de que nginx en la computadora del usuario y tener que modificar archivos, se decidió correr el servidor DoH en un contenedor docker. Se debe setear la variable de entorno DOCKER especificando el programa a utilizar para manejar contenedores.

En una primera instancia, ante la situación de ingresar AF_UNSPEC en el addrinfo hints a la hora de resolver un nombre se intentó crear un dns-message que incluya 2 preguntas, uno con el campo A y otro idéntico con el campo AAAA.

Sin embargo, al recibir la respuesta, responde con un solo tipo en el mensaje de respuesta, en nuestro caso, solo direcciones ipv6 o inclusive ninguno (solo se obtiene un SOA). Para lograr lo planteado se decidió iterar por el addrinfo de

respuesta, y en caso de de que no haya en él un elemento con una dirección ipv4, se vuelve a crear un dns-message preguntando únicamente por la dirección ipv4.

LIMITACIONES DE LA APLICACIÓN

Algunas funcionalidades del protocolo SOCKv5 no llegaron a ser implementadas para nuestra aplicación, como son el BIND y UDP ASSOCIATE. El servidor tampoco soporta conexiones UDP, usando únicamente TCP para los clientes y SCTP para los administradores.

Si bien el servidor soporta autenticación, la cantidad de usuarios que pueden existir al mismo tiempo se encuentra muy limitada, permitiendo hasta 10 usuarios únicamente, y su existencia es volátil, ya que la lista desaparece al reiniciar el servidor.

El disector de contraseñas solo soporta tres casos. Para HTTP, la autorización debe ser de tipo Basic. Cualquier otro tipo de autorización será ignorado. Para POP3, soporta tanto la autenticación mediante los comandos user y pass (donde viajan en texto plano) y mediante “auth plain”, donde el usuario y la contraseña están encodeados en Base64.

POSIBLES EXTENSIONES

Un feature que quisimos incluir pero optamos por postergar fue la persistencia de métricas y usuarios, permitiendo reanudar el estado del servidor una vez reiniciado, obviamente sin las conexiones que quedaron inconclusas cuando cayó el servidor.

Otra posible extensión sería la implementación de BIND o UDP ASSOCIATE para brindar mayor cobertura a las especificaciones del protocolo.

En lo posible, se podría ampliar el conjunto de tipos de autenticación para soportar mayor variedad de codificaciones, para más protocolos que solo POP3 y HTTP.

Finalmente, otra extensión factible sería ampliar el protocolo de administrador para modificar aún más variables. Algunas de las variables que se podrían cambiar son el tamaño del pool del proxy, la cantidad máxima de usuarios aceptados o el tamaño máximo de buffers.

CONCLUSIONES

El principal aprendizaje realizado a lo largo del desarrollo de este trabajo fue la importancia de los protocolos a la hora de implementar una aplicación que se conecte con otras, ya que establece las reglas a seguir y define un lenguaje que ambos entiendan, sin decisiones arbitrarias de cada implementación.

También destacamos el aprendizaje sobre manejo de sockets, partiendo de la experiencia limitada que teníamos con ellos de la materia Sistemas Operativos y respaldado por la teoría y conocimientos prácticos obtenidos en Protocolos de Comunicación.

Por otro lado, se aprendió a utilizar nuevo software como Wireshark, el cual fue extremadamente útil no solo a la hora de hacer el trabajo práctico especial, pero también para entender mejor los protocolos aprendidos durante la materia y el intercambio de paquetes establecido por estos.

Adicionalmente, si bien no está estrechamente vinculado con el propósito del trabajo, también presenciamos la falta de compatibilidad POSIX entre distintos sistemas operativos, ya que muchas constantes generaban errores de compilación en sistemas operativos Mac cuando eran aceptadas en Linux.

EJEMPLOS DE PRUEBA

Para probar el funcionamiento básico del proxy, se intentó acceder a una página a través de curl desde una terminal mientras el servidor corría en otra. En la terminal donde se ejecutó el comando curl, se imprimió por pantalla el HTML de la página de Google, mientras que el servidor registró el acceso del usuario “juan” con timestamp, dirección IP desde la cual accedió, puerto del cual se conectó, dirección IP a la cual quiso acceder (en este caso, curl resolvió el nombre de dominio a una IPv6) y el puerto al que accedió (por tratarse de un request HTTP, fue al puerto 80). Como la conexión resultó exitosa, se pudo observar un código de estado 0 al final de la línea.

curl -x socks5://localhost:{puerto proxy} -U juan:juan --url http://www.google.com

```
santi@santi-Lenovo-V330-15IKB:~/repos/pc-2020a-2$ ./main -u juan:juan
DEBUG: Proxy/main.c:105         2020-05-23T04:18:00Z            Esperando clientes en [::]:1080 (TCP)
DEBUG: Proxy/main.c:80          2020-05-23T04:18:00Z            Esperando clientes en 0.0.0.0:1080 (TCP)
DEBUG: Proxy/main.c:160         2020-05-23T04:18:00Z            Esperando administradores en 127.0.0.1:8080 (TCP)
2020-05-23T04:18:03Z    juan    A       127.0.0.1       35614   2800:03f0:4002:080a:0000:0000:0000:2004 80      0
```

Para poner a prueba el disector de contraseñas, se volvió a ejecutar el mismo comando pero esta vez con el flag -v para poder ver los headers enviados y el flag -u con el par usuario:contraseña a ser enviado dentro del request. Se observó entonces el header “Authorization: Basic” que contenía los datos ingresados codificados en Base64.

curl -v -u protocolos:comunicacion -x socks5://localhost:{puerto proxy} -U juan:juan --url http://www.google.com

```
santi@santi-Lenovo-V330-15IKB:~/repos/pc-2020a-2$ curl -v -u protocolos:comunicacion -x socks5://lo
* Rebuilt URL to: http://www.google.com/
*   Trying 127.0.0.1...
* TCP_NODELAY set
* SOCKS5 communication to www.google.com:80
* SOCKS5 connect to IPv6 2800:3f0:4002:808::2004 (locally resolved)
* SOCKS5 request granted.
* Connected to localhost (127.0.0.1) port 1080 (#0)
* Server auth using Basic with user 'protocolos'
> GET / HTTP/1.1
> Host: www.google.com
> Authorization: Basic cHJvdG9jb2xvczpjb211bmljYWNpb24=
> User-Agent: curl/7.58.0
> Accept: */*
>
```

Volviendo al servidor, no solo se registró el acceso de “juan” como la última vez, sino que también se expuso el usuario y contraseña que habían sido enviados bajo codificación, aclarando cuándo se envió esa autenticación, a dónde, y de qué protocolo se trata.

```
santi@santi-Lenovo-V330-15IKB:~/repos/pc-2020a-2$ ./main -u juan:juan
DEBUG: Proxy/main.c:105         2020-05-23T04:29:09Z            Esperando clientes en [::]:1080 (TCP)
DEBUG: Proxy/main.c:80          2020-05-23T04:29:09Z            Esperando clientes en 0.0.0.0:1080 (TCP)
DEBUG: Proxy/main.c:160         2020-05-23T04:29:09Z            Esperando administradores en 127.0.0.1:8080 (TCP)
2020-05-23T04:29:12Z    juan    A       127.0.0.1       35732   2800:03f0:4002:0809:0000:0000:0000:2004 80      0
2020-05-23T04:29:12Z    juan    P       HTTP    2800:3f0:4002:809::2004 80      protocolos      comunicacion
```

Para probar la disponibilidad del servidor a otras computadoras que no sean la que está corriendo la aplicación del proxy, se utilizó una máquina virtual para realizar una conexión a través del proxy a un servidor dovecot alojado en la máquina virtual mencionada. Usando el flag -v de curl, y especificando con qué usuario y contraseña acceder al servidor POP3, pudimos ver el intercambio por consola, donde el usuario se autenticó mediante AUTH PLAIN, con sus datos privados codificados en Base64.

curl -v -U juan:juan -x socks5://{IP proxy}:{puerto proxy} pop3://user:pass@{IP dovecot}

```
santi-vm@santi-vm:~$ curl -v -U juan:juan -x socks5://192.168.56.1:1080 pop3://test:1111@192.168.56.101
*   Trying 192.168.56.1:1080...
* TCP_NODELAY set
* SOCKS5 communication to 192.168.56.101:110
* SOCKS5 connect to IPv4 192.168.56.101:110 (locally resolved)
* SOCKS5 request granted.
* Connected to 192.168.56.1 (192.168.56.1) port 1080 (#0)
< +OK [XCLIENT] Dovecot (Ubuntu) ready.
> CAPA
< +OK
< CAPA
< TOP
< UIDL
< RESP-CODES
< PIPELINING
< AUTH-RESP-CODE
< STLS
< USER
< SASL PLAIN
< .
> AUTH PLAIN
< +
> AHRlc3QAMTExMQ==
< +OK Logged in.
> LIST
< +OK 0 messages:
```

De la misma manera que en el caso anterior, el usuario y la contraseña fueron extraídos con éxito, esta vez aclarando que se trataba de una conexión POP3.

```
santi@santi-Lenovo-V330-15IKB:~/repos/pc-2020a-2$ ./main -u juan:juan
DEBUG: Proxy/main.c:105         2020-05-23T04:39:44Z            Esperando clientes en [::]:1080 (TCP)
DEBUG: Proxy/main.c:80          2020-05-23T04:39:44Z            Esperando clientes en 0.0.0.0:1080 (TCP)
DEBUG: Proxy/main.c:160         2020-05-23T04:39:44Z            Esperando administradores en 127.0.0.1:8080 (TCP)
2020-05-23T04:40:03Z    juan    A       192.168.56.101  58724   192.168.56.101  110     0
2020-05-23T04:40:03Z    juan    P       POP3    192.168.56.101  110     test    1111
```

Para probar la tercer forma de autenticación soportada por el disector, se utilizó netcat para acceder al puerto donde corre dovecot y autenticarse mediante el comando user y el comando pass.

curl -v -U juan:juan -x socks5://{IP proxy}:{puerto proxy} pop3://user:pass@{IP dovecot}

```
santi-vm@santi-vm:~$ nc -X 5 -x 192.168.56.1:1080 192.168.56.101 110
+OK [XCLIENT] Dovecot (Ubuntu) ready.
user test
+OK
pass 1111
+OK Logged in.
list
+OK 0 messages:
.
quit
+OK Logging out.
```

Una vez más, se registró el acceso desde la IP de la máquina virtual y se extrajeron las contraseñas, que en este caso viajaban en texto plano en vez de estar codificadas.

```
santi@santi-Lenovo-V330-15IKB:~/repos/pc-2020a-2$ ./main -u juan:juan
DEBUG: Proxy/main.c:105         2020-05-23T04:47:28Z            Esperando clientes en [::]:1080 (TCP)
DEBUG: Proxy/main.c:80          2020-05-23T04:47:28Z            Esperando clientes en 0.0.0.0:1080 (TCP)
DEBUG: Proxy/main.c:160         2020-05-23T04:47:28Z            Esperando administradores en 127.0.0.1:8080 (TCP)
2020-05-23T04:47:33Z    ANON    A       192.168.56.101  58730   192.168.56.101  110     0
2020-05-23T04:47:38Z    ANON    P       POP3    192.168.56.101  110     test    1111
```

Para verificar que los mensajes se envían correctamente sin esperar que la información llegue completa al mismo tiempo, se utilizó stty para enviar bytes al proxy uno por uno, a medida que se escribían. La prueba pudo hacerse con Wireshark, verificando que al iniciar la captura antes de enviar el \r\n, ya se estaban leyendo nuevos bytes a medida que llegaban al servidor.

stty -icanon && nc -v -X 5 -x {IP proxy}:{puerto proxy} www.google.com 80

```
santi-vm@santi-vm:~$ stty -icanon && nc -v -X 5 -x 192.168.56.1:1080 www.google.com 80
Connection to www.google.com 80 port [tcp/http] succeeded!
GET / HTTP/1.1

HTTP/1.1 200 OK
Date: Tue, 23 Jun 2020 07:50:56 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
Server: gws
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
Set-Cookie: 1P_JAR=2020-06-23-07; expires=Thu, 23-Jul-2020 07:50:56 GMT; path=/; domain=.g
Set-Cookie: NID=204=UmEeNaooQWwp8NpE8HV5FfiJrE0GZN60s-C3j-8tvJ3pjPjzhDg67ixltANpe-6AO9Sc52
Bmb704c_6vaWSKsCcUYVCG5kCPJEJ3mg3G6jIPSo70; expires=Wed, 23-Dec-2020 07:50:56 GMT; path=/;
Accept-Ranges: none
Vary: Accept-Encoding
Transfer-Encoding: chunked

59bd
<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="es-419"><head
nt-Type"><meta content="/images/branding/googleg/1x/googleg_standard_color_128dp.png" item
KU7FHTiC8HeQ==">(function(){window.google={kEI:'YLTxXvTDCua05OUPmaaTmAc',kEXPI:'0,202123,3
```

Una vez más, el acceso se registró exitosamente

```
santi@santi-Lenovo-V330-15IKB:~/repos/pc-2020a-2$ ./main -u juan:juan
DEBUG: Proxy/main.c:105         2020-05-23T04:50:39Z            Esperando clientes en [::]:1080 (TCP)
DEBUG: Proxy/main.c:80          2020-05-23T04:50:39Z            Esperando clientes en 0.0.0.0:1080 (TCP)
DEBUG: Proxy/main.c:160         2020-05-23T04:50:39Z            Esperando administradores en 127.0.0.1:8080 (TCP)
2020-05-23T04:50:42Z    ANON    A       192.168.56.101  58732   www.google.com (2800:3f0:4002:80b::2004)        80      0
```

Para verificar que los nombres que no hayan podido ser resueltos sean notificados como tal, se probó acceder a un sitio que no existe, y efectivamente el servidor lo reflejó en su registro y en su respuesta al cliente.

```
santi@santi-Lenovo-V330-15IKB:~/repos/pc-2020a-2$ ncat --proxy 127.0.0.1:1080 --proxy-auth juan:juan --proxy-type socks5 www.estamateriasellamaprotocolosdecomunicacion.
com 80
Ncat: Error: Host unreachable.
```

```
santi@santi-Lenovo-V330-15IKB:~/repos/pc-2020a-2$ ./main -u juan:juan
DEBUG: Proxy/main.c:105         2020-05-23T04:57:22Z            Esperando clientes en [::]:1080 (TCP)
DEBUG: Proxy/main.c:80          2020-05-23T04:57:22Z            Esperando clientes en 0.0.0.0:1080 (TCP)
DEBUG: Proxy/main.c:160         2020-05-23T04:57:22Z            Esperando administradores en 127.0.0.1:8080 (TCP)
DEBUG: Proxy/socks5.c:583               2020-05-23T05:02:41Z            "www.estamateriasellamaprotocolosdecomunicacion.com" no pudo ser resuelto por DoH. Usando getadd
rinfo
2020-05-23T05:02:41Z    juan    A       127.0.0.1       36102   www.estamateriasellamaprotocolosdecomunicacion.com      80      4
```

Para verificar que el servidor escuche en limitadas interfaces, se probó establecer una IP sobre la cual escuchará. En este caso, en una IP sobre la interfaz wlp2s0. Intentando acceder al proxy con esa IP, funciona normalmente, pero si se lo referencia con una IP por fuera de esa interfaz (por ejemplo, 127.0.0.1, que es su IP en la interfaz loopback) falla.

Lo mismo ocurre si se intenta acceder al servidor con una IPv6 mientras el mismo está bindeado a una IPv4.

```
santi@santi-Lenovo-V330-15IKB:~/repos/pc-2020a-2$ curl -v -u protocolos:comunicacion -x socks5://192.168.0.7:1080 -U juan:juan --url http://www.facebook.com
* Rebuilt URL to: http://www.facebook.com/
*   Trying 192.168.0.7...
* TCP_NODELAY set
* SOCKS5 communication to www.facebook.com:80
* SOCKS5 connect to IPv6 2a03:2880:f110:83:face:b00c:0:25de (locally resolved)
* SOCKS5 request granted.
* Connected to 192.168.0.7 (192.168.0.7) port 1080 (#0)
* Server auth using Basic with user 'protocolos'
> GET / HTTP/1.1
> Host: www.facebook.com
> Authorization: Basic cHJvdG9jb2xvczpjb211bmljYWNpb24=
> User-Agent: curl/7.58.0
> Accept: */*
>
< HTTP/1.1 302 Found
< Location: https://www.facebook.com/
< Content-Type: text/html; charset="utf-8"
< X-FB-Debug: 9YvVv73yzNjXx4qMOY2mzyyI0EpYdFZ7kqIr+HIntGiG9AF66TVVyyG/v+EouB8JyAh4qPaUoVisPzpXra/lHw==
< Date: Tue, 23 Jun 2020 08:06:07 GMT
< Alt-Svc: h3-29=":443"; ma=3600,h3-27=":443"; ma=3600
< Connection: keep-alive
< Content-Length: 0
<
* Connection #0 to host 192.168.0.7 left intact
santi@santi-Lenovo-V330-15IKB:~/repos/pc-2020a-2$ curl -v -u protocolos:comunicacion -x socks5://127.0.0.1:1080 -U juan:juan --url http://www.facebook.com
* Rebuilt URL to: http://www.facebook.com/
*   Trying 127.0.0.1...
* TCP_NODELAY set
* connect to 127.0.0.1 port 1080 failed: Connection refused
* Failed to connect to 127.0.0.1 port 1080: Connection refused
* Closing connection 0
curl: (7) Failed to connect to 127.0.0.1 port 1080: Connection refused
```

```
santi@santi-Lenovo-V330-15IKB:~/repos/pc-2020a-2$ curl -v -u protocolos:comunicacion -x socks5://[::1]:1080 -U juan:juan --url http://www.facebook.com
* Rebuilt URL to: http://www.facebook.com/
*   Trying ::1...
* TCP_NODELAY set
* connect to ::1 port 1080 failed: Connection refused
* Failed to connect to ::1 port 1080: Connection refused
* Closing connection 0
curl: (7) Failed to connect to ::1 port 1080: Connection refused
santi@santi-Lenovo-V330-15IKB:~/repos/pc-2020a-2$
```

Similarmente, si el servidor escucha sobre una interfaz IPv6 (en este caso, en la IP ::1), no se podrá acceder mediante IPs que no pertenezcan a loopback de IPv6. Ni siquiera con la IP de loopback en IPv4.

```
santi@santi-Lenovo-V330-15IKB:~/repos/pc-2020a-2$ curl -v -u protocolos:comunicacion -x socks5://[::1]:1080 -U juan:juan --url http://www.facebook.com
* Rebuilt URL to: http://www.facebook.com/
*   Trying ::1...
* TCP_NODELAY set
* SOCKS5 communication to www.facebook.com:80
* SOCKS5 connect to IPv6 2a03:2880:f110:83:face:b00c:0:25de (locally resolved)
* SOCKS5 request granted.
* Connected to ::1 (::1) port 1080 (#0)
* Server auth using Basic with user 'protocolos'
> GET / HTTP/1.1
> Host: www.facebook.com
> Authorization: Basic cHJvdG9jb2xvczpjb211bmljYWNpb24=
> User-Agent: curl/7.58.0
> Accept: */*
>
< HTTP/1.1 302 Found
< Location: https://www.facebook.com/
< Content-Type: text/html; charset="utf-8"
< X-FB-Debug: LfsIG+tqyDHhhWNzd1XKYQrWFe4/V2Px/Okj/JFs47VG4r/Uh6Xl7mGuEO9KubN2fji7c1pvDgQFvwLG7j4GXw==
< Date: Tue, 23 Jun 2020 08:11:06 GMT
< Alt-Svc: h3-29=":443"; ma=3600,h3-27=":443"; ma=3600
< Connection: keep-alive
< Content-Length: 0
<
* Connection #0 to host ::1 left intact
santi@santi-Lenovo-V330-15IKB:~/repos/pc-2020a-2$ curl -v -u protocolos:comunicacion -x socks5://192.168.0.7:1080 -U juan:juan --url http://www.facebook.com
* Rebuilt URL to: http://www.facebook.com/
*   Trying 192.168.0.7...
* TCP_NODELAY set
* connect to 192.168.0.7 port 1080 failed: Connection refused
* Failed to connect to 192.168.0.7 port 1080: Connection refused
* Closing connection 0
curl: (7) Failed to connect to 192.168.0.7 port 1080: Connection refused
```

```
santi@santi-Lenovo-V330-15IKB:~/repos/pc-2020a-2$ curl -v -u protocolos:comunicacion -x socks5://127.0.0.1:1080 -U juan:juan --url http://www.facebook.com
* Rebuilt URL to: http://www.facebook.com/
*   Trying 127.0.0.1...
* TCP_NODELAY set
* connect to 127.0.0.1 port 1080 failed: Connection refused
* Failed to connect to 127.0.0.1 port 1080: Connection refused
* Closing connection 0
curl: (7) Failed to connect to 127.0.0.1 port 1080: Connection refused
```

Para verificar que las direcciones se resuelvan con DNS Over HTTP, se puede probar accediendo a varios sitios y ver que no se imprime el mensaje de

advertencia que indica cuando DOH falló a la hora de resolver un nombre. Además, puede comprobarse que resuelve tanto a IPv6 como IPv4.

```
santi@santi-Lenovo-V330-15IKB:~/repos/pc-2020a-2$ ./main -u juan:juan
DEBUG: Proxy/main.c:105         2020-05-23T05:16:30Z            Esperando clientes en [::]:1080 (TCP)
DEBUG: Proxy/main.c:80          2020-05-23T05:16:30Z            Esperando clientes en 0.0.0.0:1080 (TCP)
DEBUG: Proxy/main.c:160         2020-05-23T05:16:30Z            Esperando administradores en 127.0.0.1:8080 (TCP)
2020-05-23T05:16:59Z    juan    A       127.0.0.1       36972   itba.edu.ar (54.233.177.225)    80      0
2020-05-23T05:17:13Z    juan    A       127.0.0.1       36988   www.facebook.com (2a03:2880:f110:83:face:b00c:0:25de)   80      0
```

Para verificar que el servidor pueda soportar más de 500 clientes concurrentes, se desarrolló un script para automatizar un conjunto de usuarios que intentan conectarse a un sitio a través del servidor. El script consistió en crear varios procesos donde se establece una conexión con el servidor mediante netcat, manteniendo así una conexión abierta mientras envían un request HTTP y el proxy la envía al sitio a donde se conectó. Eventualmente se logró tener 505 usuarios conectados al mismo tiempo enviando información. Todo usuario que intentó conectarse mientras había 505 activos fueron rechazados.

```
santi@santi-Lenovo-V330-15IKB:~/repos/pc-2020a-2$ ./manager -u juan:juan -m
Conexiones históricas:   505
Conexiones concurrentes:   505
Bytes transferidos:    22463043
```

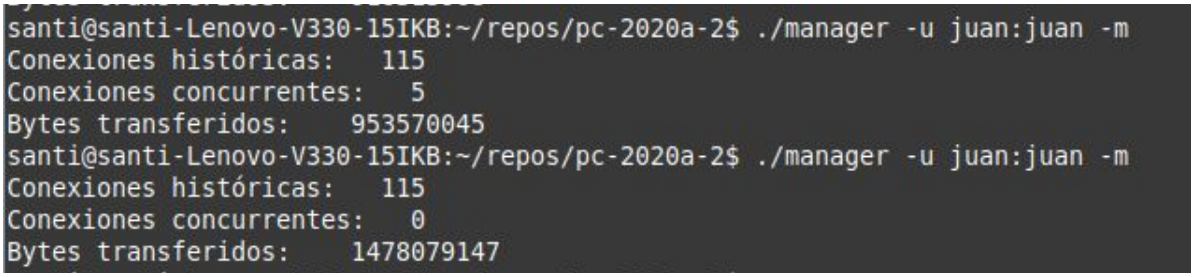
```
2020-05-23T06:02:14Z    juan    A       127.0.0.1       59544   www.google.com (2800:3f0:4002:80a::2004)        80      0
2020-05-23T06:02:14Z    juan    A       127.0.0.1       59428   www.google.com (2800:3f0:4002:80b::2004)        80      0
2020-05-23T06:02:14Z    juan    A       127.0.0.1       59748   www.google.com (2800:3f0:4002:80a::2004)        80      0
2020-05-23T06:02:14Z    juan    A       127.0.0.1       59656   www.google.com (2800:3f0:4002:809::2004)        80      0
2020-05-23T06:02:14Z    juan    A       127.0.0.1       60154   www.google.com (2800:3f0:4002:80a::2004)        80      0
2020-05-23T06:02:14Z    juan    A       127.0.0.1       59658   www.google.com (2800:3f0:4002:802::2004)        80      0
2020-05-23T06:02:14Z    juan    A       127.0.0.1       59654   www.google.com (2800:3f0:4002:808::2004)        80      0
2020-05-23T06:02:15Z    juan    A       127.0.0.1       59590   www.google.com (2800:3f0:4002:808::2004)        80      0
2020-05-23T06:02:15Z    juan    A       127.0.0.1       59586   www.google.com (2800:3f0:4002:80d::2004)        80      0
2020-05-23T06:02:15Z    juan    A       127.0.0.1       59650   www.google.com (2800:3f0:4002:802::2004)        80      0
2020-05-23T06:02:15Z    juan    A       127.0.0.1       59526   www.google.com (2800:3f0:4002:800::2004)        80      0
2020-05-23T06:02:15Z    juan    A       127.0.0.1       59452   www.google.com (2800:3f0:4002:80a::2004)        80      0
2020-05-23T06:02:15Z    juan    A       127.0.0.1       59530   www.google.com (2800:3f0:4002:809::2004)        80      0
2020-05-23T06:02:15Z    juan    A       127.0.0.1       59528   www.google.com (2800:3f0:4002:808::2004)        80      0
2020-05-23T06:02:15Z    juan    A       127.0.0.1       59438   www.google.com (2800:3f0:4002:80a::2004)        80      0
2020-05-23T06:02:15Z    juan    A       127.0.0.1       59520   www.google.com (2800:3f0:4002:80d::2004)        80      0
2020-05-23T06:02:16Z    juan    A       127.0.0.1       59774   www.google.com (2800:3f0:4002:80d::2004)        80      0
2020-05-23T06:02:16Z    juan    A       127.0.0.1       59724   www.google.com (2800:3f0:4002:802::2004)        80      0
2020-05-23T06:02:16Z    juan    A       127.0.0.1       59912   www.google.com (2800:3f0:4002:80b::2004)        80      0
2020-05-23T06:02:17Z    juan    A       127.0.0.1       59550   www.google.com (2800:3f0:4002:80a::2004)        80      0
2020-05-23T06:02:17Z    juan    A       127.0.0.1       59472   www.google.com (2800:3f0:4002:80a::2004)        80      0
2020-05-23T06:02:17Z    juan    A       127.0.0.1       59630   www.google.com (2800:3f0:4002:80a::2004)        80      0
2020-05-23T06:02:17Z    juan    A       127.0.0.1       59562   www.google.com (2800:3f0:4002:80a::2004)        80      0
2020-05-23T06:02:17Z    juan    A       127.0.0.1       59400   www.google.com (2800:3f0:4002:80d::2004)        80      0
2020-05-23T06:02:17Z    juan    A       127.0.0.1       59616   www.google.com (2800:3f0:4002:80a::2004)        80      0
2020-05-23T06:02:18Z    juan    A       127.0.0.1       59622   www.google.com (2800:3f0:4002:802::2004)        80      0
2020-05-23T06:02:18Z    juan    A       127.0.0.1       59618   www.google.com (2800:3f0:4002:808::2004)        80      0
2020-05-23T06:02:18Z    juan    A       127.0.0.1       59638   www.google.com (2800:3f0:4002:809::2004)        80      0
2020-05-23T06:02:18Z    juan    A       127.0.0.1       59534   www.google.com (2800:3f0:4002:80d::2004)        80      0
2020-05-23T06:02:18Z    juan    A       127.0.0.1       59532   www.google.com (2800:3f0:4002:80a::2004)        80      0
2020-05-23T06:02:18Z    juan    A       127.0.0.1       59470   www.google.com (2800:3f0:4002:808::2004)        80      0
```

```
DEBUG: Proxy/socks5.c:140               2020-05-23T05:52:16Z            Conexión rechazada. Se superó el límite de clientes 505.

DEBUG: Proxy/socks5.c:140               2020-05-23T05:52:16Z            Conexión rechazada. Se superó el límite de clientes 505.

DEBUG: Proxy/socks5.c:140               2020-05-23T05:52:16Z            Conexión rechazada. Se superó el límite de clientes 505.
```

De forma similar, para verificar que el servidor pueda soportar transporte de muchos datos para una conexión, se desarrolló un script donde 50 usuarios intentan descargar al mismo tiempo un archivo de 100MB. Podemos observar que desde que se inició el programa (cuando esos 5 usuarios eran los únicos conectados), se transfirieron aproximadamente 500 MB por el proxy.



GUÍA DE INSTALACIÓN

Es requisito tener docker o similar instalado y definir las variables de entorno CC y DOCKER.

Para clonar el repositorio y acceder a la carpeta se deben usar los comandos:
git clone git@bitbucket.org:itba/pc-2020a-2.git
cd pc2020a-2.git

Luego se ingresa a la carpeta src donde se compila y linkeditan los ejecutables main (servidor proxy) y manager (cliente administrador):
make all

Para correr el servidor se ejecuta el main:
./main

El servidor DoH se corre por separado. Para armar la imagen doh-nginx y descargar el contenido de nginx se usa el comando:
make doh-build

Para luego correr el contenedor doh-server se usa el comando:
make doh-start

Si se desea frenar la ejecución del doh-server se usa el comando:
make doh-stop

Para ejecutar manager (cliente admin):
./manager

Para correr los tests se usa el comando:
make tests

Para borrar todos los archivos ejecutables y temporales se usa el comando:
make clean

Para extraer del tar:
tar -xf pc2020a-2.tar.gz

INSTRUCCIONES PARA LA CONFIGURACIÓN

Para correr y configurar el proxy se utiliza el comando ./main seguido de los flags:

-v para la ver la versión.

-h para imprimir la ayuda.

-l <SOCKS dirección> para setear la dirección donde servirá el proxy SOCKS.

-L <manager dirección> para setear dirección donde servirá el servicio de management.

-p <SOCKS puerto> para setear el puerto entrante de conexiones SOCKS

-P <manager puerto> para setear el puerto entrante de conexiones manager

-u <name>:<pass> para poner el usuario y contraseña de usuario que puede usar el proxy. Máximo 10.

--doh-ip <ip> para configurar la IP del DoH server.

--doh-port <puerto> para configurar el puerto del DoH server.

--doh-host <host> para configurar el header host del DoH server.

--doh-path <path> para configurar el path del request DoH.

--doh-query <query> para configurar la query string del request DoH.

Para correr y configurar el manager se utiliza el comando ./manager seguido de las flags:

-u <usuario>:<contraseña> para poder pasar la autorización del proxy ingresando con un usuario registrado y su contraseña. Esta flag es obligatoria.

-U para recibir la lista de usuarios registrados.

-a <usuario>:<contraseña> para agregar un usuario y su contraseña al registro, máximo 10.

-v para la ver la versión.

-h para imprimir la ayuda.

-l <SOCKS dirección> para setear la dirección donde sirve el proxy SOCKS.

-p <SOCKS puerto> para setear el puerto entrante del proxy SOCKS.

-m para obtener las métricas, las cuales incluyen la cantidad de bytes transferidos, las conexiones concurrentes y las conexiones históricas.

-s <nuevo tamaño> para setear la cantidad de clientes que el proxy acepta, máximo 505 .

Las flags de comando (-a, -U, -s y -m) se pueden utilizar una a la vez.

EJEMPLOS DE CONFIGURACIÓN Y MONITOREO

Para el cliente administrador, se realizaron las siguientes pruebas:
Primero obtenemos la lista de comandos haciendo **./manager -h**

```
santi@santi-Lenovo-V330-15IKB:~/repos/pc-2020a-2$ ./manager -h
Uso: ./manager [OPCIÓN]...

   -h               Imprime la ayuda y termina.
   -l <SOCKS addr>  Dirección donde servirá el proxy SOCKS.
   -p <SOCKS port>  Puerto entrante conexiones SOCKS.
   -u <name>:<pass> Usuario y contraseña de usuario que quiere usar el manager.
   -a <name>:<pass> Agregar un usuario y contraseña.
   -m               Obtener métricas sobre el funcionamiento del servidor.
   -U               Listar usuarios del proxy.
   -s <new size>    Cambiar cantidad máxima de usuarios concurrentes. Máximo 505 clientes.
   -v               Imprime información sobre la versión y termina.
```

Para toda operación excepto -h y -v, debemos autenticarnos utilizando el flag -u.

Podemos probar cambiando la interfaz y puerto donde el proxy espera las conexiones de manager (eligiendo valores distintos a los predeterminados) y acceder al mismo desde manager con los flags -l y -p. Podemos usar además el flag -U para pedir la lista de usuarios habilitados.

./main -L 192.168.0.7 -P 9090 -u juan:juan
./manager -u juan:juan -U -l 192.168.0.7 -p 9090

También funciona en IPv6:

./main -L 2800:810:430:83fd:306c:1fe2:2bcc:1f6f -P 9090 -u juan:juan
./manager -u juan:juan -U -l 2800:810:430:83fd:306c:1fe2:2bcc:1f6f -p 9090

```
santi@santi-Lenovo-V330-15IKB:~/repos/pc-2020a-2$ ./manager -u juan:juan -U -l 2800:810:430:83fd:306c:1fe2:2cbc:1f6f -p 9090
Usuarios actuales:
juan
```

Volviendo a lanzar el server en la dirección y puerto default para simplificar los comandos de aquí en más, podemos añadir usuarios al servidor con el comando -a. Sólo se permiten 10 usuarios registrados en todo momento dado.

./manager -u juan:juan -a marcelo:garberoglio

```
santi@santi-Lenovo-V330-15IKB:~/repos/pc-2020a-2$ ./manager -u juan:juan -a marcelo:garberoglio
Nuevo usuario agregado: marcelo
santi@santi-Lenovo-V330-15IKB:~/repos/pc-2020a-2$ ./manager -u juan:juan -a ariel:so
Nuevo usuario agregado: ariel
santi@santi-Lenovo-V330-15IKB:~/repos/pc-2020a-2$ ./manager -u juan:juan -a franco:poo
Nuevo usuario agregado: franco
santi@santi-Lenovo-V330-15IKB:~/repos/pc-2020a-2$ ./manager -u juan:juan -a juanma:sotuyo
Nuevo usuario agregado: juanma
santi@santi-Lenovo-V330-15IKB:~/repos/pc-2020a-2$ ./manager -u juan:juan -a santiago:valles
Nuevo usuario agregado: santiago
santi@santi-Lenovo-V330-15IKB:~/repos/pc-2020a-2$ ./manager -u juan:juan -a horacio:so
Nuevo usuario agregado: horacio
santi@santi-Lenovo-V330-15IKB:~/repos/pc-2020a-2$ ./manager -u juan:juan -a santiago:terenziani
Nuevo usuario agregado: santiago
santi@santi-Lenovo-V330-15IKB:~/repos/pc-2020a-2$ ./manager -u juan:juan -a enrique:tawara
Nuevo usuario agregado: enrique
santi@santi-Lenovo-V330-15IKB:~/repos/pc-2020a-2$ ./manager -u juan:juan -a sofia:picasso
Nuevo usuario agregado: sofia
santi@santi-Lenovo-V330-15IKB:~/repos/pc-2020a-2$ ./manager -u juan:juan -a cosme:fulanito
No se pudo crear el usuario. Es posible que se haya superado el máximo de usuarios permitidos.
Error: Cerrando conexión
```

Podemos revisar la lista en cualquier momento con -U

./manager -u juan:juan -U

```
santi@santi-Lenovo-V330-15IKB:~/repos/pc-2020a-2$ ./manager -u juan:juan -U
Usuarios actuales:
juan
marcelo
ariel
franco
juanma
santiago
horacio
santiago
enrique
sofia
```

Ahora podemos usar alguno de nuestros nuevos usuarios para, por ejemplo, solicitar las métricas del proxy. Podemos observar que si ejecutamos el mismo comando más tarde, las métricas reflejarán el estado en ese momento.

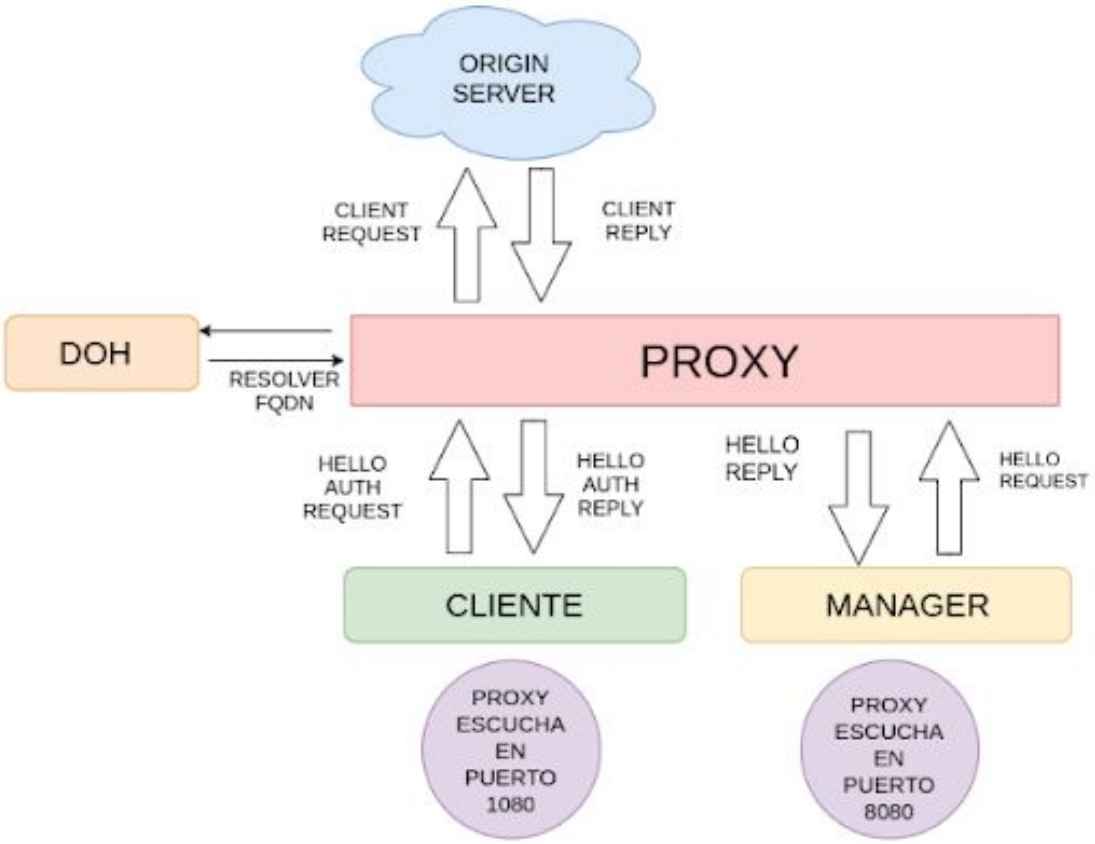
```
santi@santi-Lenovo-V330-15IKB:~/repos/pc-2020a-2$ ./manager -u marcelo:garberoglio -m
Conexiones históricas:   31
Conexiones concurrentes:   24
Bytes transferidos:    363905
```

Finalmente, podemos modificar el límite de usuarios con -s. En caso de que el nuevo límite sea menor a la cantidad de usuarios actuales, no se aceptarán nuevas conexiones hasta que la cantidad de usuarios activos caiga por debajo de ese límite.

```
santi@santi-Lenovo-V330-15IKB:~/repos/pc-2020a-2$ ./manager -u marcelo:garberoglio -s 3
Nueva cantidad máxima de clientes simultáneos: 3
```

```
2020-05-23T07:54:03Z    ANON    A       ::1     49434   update.googleapis.com (2800:3f0:4002:80c::2003) 443     0
2020-05-23T07:54:03Z    ANON    A       ::1     49450   storage.googleapis.com (2800:3f0:4002:803::2010)        80      0
2020-05-23T07:54:04Z    ANON    A       ::1     49466   redirector.gvt1.com (2800:3f0:4002:80d::200e)   80      0
2020-05-23T07:54:04Z    ANON    A       ::1     49482   r7---sn-uxaxjxougv-x1x6.gvt1.com (2800:810:203:2::12)   80      0
2020-05-23T07:54:12Z    ANON    A       ::1     49498   safebrowsing.googleapis.com (2800:3f0:4002:809::200a)   443     0
2020-05-23T07:58:23Z    marcelo A       127.0.0.1       47156   www.facebook.com (2a03:2880:f110:83:face:b00c:0:25de)   80      0
2020-05-23T07:58:31Z    marcelo A       127.0.0.1       47174   www.facebook.com (2a03:2880:f110:83:face:b00c:0:25de)   80      0
2020-05-23T07:58:33Z    marcelo A       127.0.0.1       47190   www.facebook.com (2a03:2880:f110:83:face:b00c:0:25de)   80      0
2020-05-23T07:58:34Z    marcelo A       127.0.0.1       47206   www.facebook.com (2a03:2880:f110:83:face:b00c:0:25de)   80      0
DEBUG: Proxy/socks5.c:140               2020-05-23T07:58:44Z            Conexión rechazada. Se superó el límite de clientes 3.
```

DOCUMENTO DE DISEÑO DEL PROYECTO



Centro del directorio src, en la carpeta Proxy del proyecto se encuentra todo lo relacionado con el funcionamiento del servidor SOCKSv5, incluyendo los archivos para atender conexiones de manager y realizar resolución de nombres con DoH.

Dentro de la carpeta Manager se encuentran los archivos de código que conforman la aplicación cliente que permite obtener métricas y cambiar la configuración del servidor en tiempo de ejecución.

doh-server contiene la configuración para levantar el servidor Nginx que resuelve los requests DNS.

Finalmente, el directorio Tests contiene archivos que prueban el correcto funcionamiento de la resolución de nombres y de los parsers que intervienen en la misma. Dentro de Tests, la carpeta Experiment Scripts contiene los dos scripts mencionados anteriormente para poner a prueba el servidor SOCKSv5 cuando hay muchos clientes conectados al mismo tiempo.

REFERENCIAS

Código utilizado para la codificación de Base64:

Source:
https://nachtimwald.com/2017/11/18/base64-encode-and-decode-in-c/

Licencia:

“All code posted here which is created by me (John Schember) is licensed under the MIT License unless otherwise specified."

https://nachtimwald.com/legal/

Las licencias MIT son licencias OSI.
https://opensource.org/license

RFC referenciados:

Domain Names: 1035
https://tools.ietf.org/html/rfc1035

Protocolo Socksv5: 1928
https://tools.ietf.org/html/rfc1928

Autenticación en protocolo Socksv5: 1929
https://tools.ietf.org/html/rfc1929

DNS Queries over HTTPS (DoH): 8484
https://tools.ietf.org/html/rfc8484