1. Índice

2. Descripción detallada de los protocolos y aplicaciones desarrolladas.

    Se desarrolló primero un servidor que funciona como Proxy SOCKS5. Esto significa que un cliente puede conectarse a un servidor remoto, utilizando nuestro servidor como intermediario. Así, el servidor remoto verá que el que se conecta es nuestro servidor y no el cliente. Así también, la conexión saliente del cliente será a nuestro servidor y no al servidor remoto.

    Al mismo, se le agregaron funcionalidades adicionales; recolección sobre estadísticas de uso, configuración de distintas variables, y la implementación de un "disector" de credenciales POP3. Esto significa que cualquier cliente que decida conectarse a un servidor POP3 mediante nuestro proxy, dejará en el mismo una copia de sus credenciales.

    Luego, se desarrolló un cliente "Administrador" (o "Admin") que se conecta a nuestro servidor por otro puerto y con otro método de certificación. Este Admin sirve, no para usar el servidor como un proxy, sino para poder editar y monitorear el mismo. Para esta comunicación, se desarrolló nuestro propio Protocolo de Comunicación, llamado SSEMD: Socks Server Editor Monitor & Dissector, el cual está descrito en el RFC adjunto en este proyecto.

3. Problemas encontrados durante el diseño y la implementación.

    El servidor pasó por varias etapas.

    En un principio se tenía un servidor TCP el cual hacía echo de lo que el cliente le mandaba. Esto se usó solamente como para no "iniciar con un pizarrón en blanco" y construir lo que posteriormente seria nuestro servidor en base a ello. Al terminar el proyecto, dudo que quede una línea de lo que era eso.

    El primer objetivo fue que el cliente se conecte, de manera bloqueante, al Servidor, y que éste se conecte a una página fija de internet, para que el Servidor automáticamente le mande la respuesta al cliente, haciendo así un proxy transparente.

    El primer problema grande fue hacer que el Servidor permita más de una conexión a la vez, y que estas puedan ser por IPv4 o IPv6.

    Otro gran problema fue hacerlo no bloqueante, que luego de un refactoreo total al uso del select(), se pudo lograr.

    Luego, para que no sea un proxy a una IP fija, se debió darle al servidor las herramientas para poder conectarse utilizando el Protocolo de Comunicación SOCKS5. Desde ahí se pudo obtener la IP variable que pedía el Cliente y no una cualquiera.

    Al terminar eso, se pudo empezar a implementar la estructura de autenticación siguiendo los lineamientos del RFC 1929.

El siguiente objetivo fue crear el cliente Admin, que se pueda conectar al servidor por otro puerto, y que el servidor maneje sus pedidos y le responda lo apropiado. Todo siguiendo los lineamientos de nuestro RFC propuesto.

Al principio el Admin mandaba un request igual a un get http, entonces era tratado como otro client, pero se podía conectar. Luego, mandaba un mensaje fijo del estilo de nuestro RFC. En este momento se tenía que darle las herramientas al servidor para que pueda comunicarse usando este nuevo Protocolo de Comunicación, que no fue demasiado complicado ya que era un parser similar al ya implementado para SOCKS5.

Luego se implementó una estructura de parámetros, inspirado por los archivos args.c y args.h provistos por la cátedra, para que el mensaje que envía el Admin no sea fijo, sino maleable.

Al momento de querer procesar la respuesta para el Admin, se tuvo que modificar un poco la estructura del Servidor, ya que ahora muchos de los parámetros que antes se consideraban fijos, ahora eran variables y configurables por el Admin (buffsize, etc.).

La implementación de este intercambio de mensajes resultó problemático, ya que ambos lados debían escribir un mensajes en uint8_t para que del otro lado sea nuevamente convertido a algo legible y tratable por humanos.

Más de una vez en este proceso, el servidor debió sufrir refactoreos grandes a su código, ya que nos quedaban archivos demasiado grandes y poco modularizados o porque se lo estaba encarando medio mal. (?¿)

Luego se debió implementar el disector de contraseñas, que trajo problemas:.

Otro problema encontrado cerca del final del proyecto fue que siempre se tomaban como ejemplo y uso, casos ideales. Lo que llevó a mas de un día de implementar, responder y avisar los casos de error. (Estaban bien señalados, solo que al toparse con un problema el servidor, cliente y/o admin quedaban en un estado inutilizable). Esto incluye memory leaks, warnings menores ignorados por demasiado tiempo y algún que otro problema menor pateado para más adelante

Otro gran problema encontrado recién al momento de escribir este informe, fue la falta de ir llenando el informe mientras se avanzaba con el proyecto. Se debería haber agregado por lo menos 1-2 frases de los problemas encontrados de ese día.

4. Limitaciones de la aplicación

Una primera limitación que tiene es que no puede manejar a más de un Admin al mismo tiempo. De todas formas, esto fue una decisión tomada conscientemente, ya que al ser una comunicación del tipo Request-Response, es demasiado rápida como para que en un caso de uso real, 2 administradores se pisen (en cual caso el segundo será debidamente notificado que el servidor está ocupado atendiendo otra consulta y que lo

intente nuevamente). Se decidió esto por sobre el contrario que sería darle posiblemente un dato erróneo a uno de los administradores.

El servidor permite hasta 10 usuarios, otra decisión tomada conscientemente, porque con 10 es suficiente para ver y probar toda la potencia del programa.

El servidor permite hasta 500 conexiones SOCKS5. Esto es porque se necesitan 2 File Descriptors por cada conexión, y linux tiene un máximo de 1024 File Descriptors, y se necesitan algunos para los sockets pasivos y para el manejo del Admin. Aún así, nos sobran unos pocos File Descriptors, pero se decidió que dejarlo en 500 no era mala idea.

5. Posibles extensiones

   El servidor:
   -Extender su disector, para que registre credenciales de http o de cualquier otro protocolo que no esté encriptado.
   -Guardar dichas credenciales en un archivo para que no se pierdan, o en alguna estructura de datos para que sean accesibles por el Admin.
   -Usuarios y contraseñas, estadísticas y configuraciones guardadas en un archivo para que no se pierdan al reiniciar el servidor.
   -Una “lista negra” de usuarios para éstos que no puedan utilizar el servidor, ya sea por IP o por [user:password] y que no puedan ser agregados nuevamente.
   -Que el servidor pueda atender a varios Admin al mismo tiempo.
   -El Admin podría ser no bloqueante.
   -El Admin podría ser un usuario registrado en el servidor, más que un token.

6. Conclusiones

   Fue un trabajo muy extenso, de los más complicados de la carrera hasta ahora. Pero un lindo desafío como para distinguirnos del resto de los mortales que limitan su uso de la tecnología en el uso cotidiano y constante de Instagram.

7. Ejemplos de prueba

8. Guía de instalación detallada y precisa. No es necesario desarrollar un programa instalador.

9. Instrucciones para la configuración.

10. Ejemplos de configuración y monitoreo.

11. Documento de diseño del proyecto (que ayuden a entender la arquitectura de la aplicación).