Protocolos de Comunicación (72.07)

Trabajo Práctico Especial - 2do Cuatrimestre 2016
XMPP-Proxy

Integrantes

Vera, Juan Sebastián	*53852*
Aagaard, Jeremias	*53219*
Guido, Sebastian Ezequiel	*54432*

Repositorio Bitbucket

pc-2016b-06

Último Commit

Índice

Índice 2
Protocolos y Aplicaciones Desarrolladas 4
Objetivo **4**
Especificaciones técnicas **4**
Inicio de la comunicación **5**
Silenciar / Bloquear usuarios **6**
Conversor l33t **6**
Métricas **6**
Accesos 6
Bytes enviados 7
Bytes recibidos 7
Mensajes bloqueados 7
Caracteres convertidos 7
Protocolo de administrador **7**
HELP 7
LOG 7
LEET_ON 8
LEET_OFF 8
LOG_OUT 8
BLOCK 8
UNBLOCK 8
MULTIPLEX 8
UNPLEX 8
PASSCHANGE 9
ACCESSES 9
BLOCKED 9
BYTES_SENT 9
BYTES_RECEIVED 9
CHARACTERS_CONVERTED 9
Registros de acceso **10**
Problemas Encontrados 11
Conversión de caracteres **11**
Convivencia de los protocolos **11**
Identificación de conexión **11**

Limitaciones de la Aplicación 15

Posibles Extensiones 14

Métricas 14

Manejo de errores 14

Administrador 14

Conclusiones 15

Ejemplos de Prueba 16

Guía de Instalación y Configuración 18

Test Plan 18

Run 18

Ejemplos de Configuración y Monitoreo 19

Documento de Diseño del Proyecyo 20

Bibliografía 21

Protocolos y Aplicaciones Desarrolladas

Objetivo

Con el siguiente trabajo práctico se busca implementar un servidor proxy para el Protocolo Extensible de Mensajería y Comunicación de Presencia (XMPP por sus siglas en inglés *Extensible Messaging and Presence Protocol).* El mismo tiene que poder ser usado por clientes XMPP para el envío y recepción de mensajes de chat.

Además de las funcionalidades propias del protocolo XMPP, el proxy provee herramientas adicionales, como lo son el bloqueo de mensajes provenientes o hacia determinados usuarios, o la conversión de mensajes (según el método *l33t*).

Para poder cumplir con lo antedicho se propone recrear el escenario (provisto por la imagen 1) en el que los clientes, en vez de conectarse directamente al servidor, se conectan al proxy, quien luego se encarga de realizar la conección con el servidor XMPP.

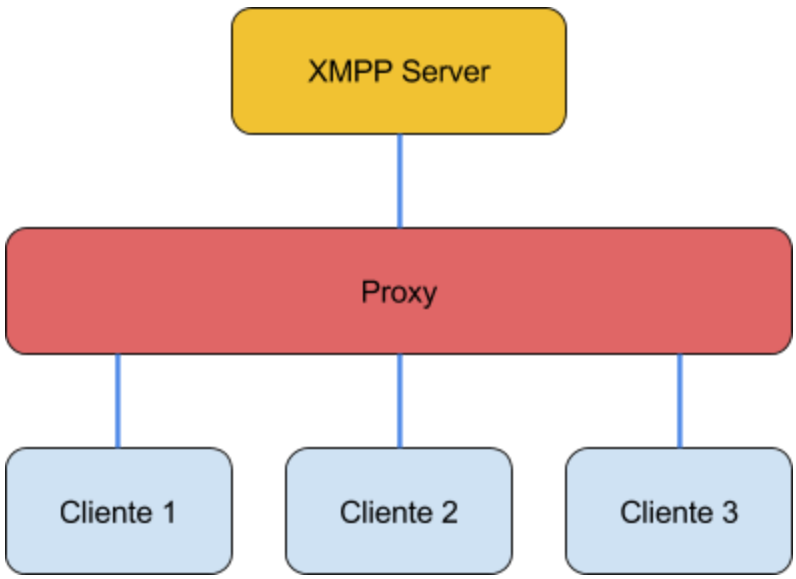

Imagen 1: Representación del escenario recreado

Especificaciones técnicas

Para lograr el escenario deseado, se utilizó de manera local el servidor XMPP Prosody. El mismo fue elegido por su fácil configuración y rápida adaptación.

A su vez, se eligió como cliente, el provisto por la aplicación Pidgin. De igual manera, fue elegido por su fácil configuración, su interfaz amigable y por la posibilidad que brinda de generar usuarios directamente en el servidor.

```
VirtualHost "protos-tpe"

admins = { "jjsebasv@protos-tpe" }
authentication = “internal_plain”;

allow_registration = true;
modules_enabled = {
                "roster";
                “register”;
                “admin_adhoc”;
                “saslauth”;
};

log = {
        -- Log files (change 'info' to 'debug' for debug logs):
        info = "/var/log/prosody/prosody-test.log";
        error = "/var/log/prosody/prosody-test.err";
        -- Syslog:
        { levels = { "error" }; to = "syslog";  };
};

c2s_require_encryption = false
c2s_ports = { 5228 }

Component "proxy.example.com" "proxy65"
proxy65_ports = { 5225 }
proxy65_interfaces = { "127.0.0.1" }
```

Cuadro 1: configuración del server

Inicio de la comunicación

Es así que cuando un cliente quiere conectarse al servidor, lo que en realidad sucede es que se conecta al proxy. Esto es transparente para el cliente, ya que el proxy se encarga de realizar el paso de información (junto con los posibles bloqueos y conversiones), desde el *handshake* hasta el mismo pasaje de mensajes.

El proxy solicita que todas las conecciones sean planas y no codificadas, para poder ‘reconocer con quién está hablando’. De esta manera, una vez que el proxy ‘conoce’ al cliente, puede simular una conección directa al servidor.

Para poder mantener una única conexión entre el cliente y el servidor, el proxy, a la hora de recibir una *Selection Key* aceptable, crea y guarda el canal que conecta al cliente con el proxy como *attachment* de la *key*, para poder acceder luego a ese canal. De igual modo guarda y crea el canal por el cual se comunicará con el servidor.

Silenciar / Bloquear usuarios

En cualquier momento, desde que el proxy fue iniciado, se pueden agregar usuarios a la lista de usuarios bloqueados.

Estos usuarios no podrán enviar ni recibir mensajes mientras sigan en esta lista.

Para determinar si un mensaje debe ser enviado, previamente se consulta si la stanza del mensaje contiene tanto en el *to* como en el *from* alguno de los usuarios de la lista. De ser así, el mensaje es simplemente descartado.

Conversor l33t

Existe la posibilidad de convertir los mensajes salientes según el modelo leet que transforma de la siguiente manera los caracteres:

a → 4 (cuatro)
e → 3 (tres)
i → 1 (uno)
o → 0 (cero)
c → < (menor)

Cuadro 2: Manera en la cual el conversor transforma los caracteres de los mensajes salientes

Para poder realizar lo antedicho, se creó un *Conversor,* por el que pasan todos los mensajes, que obtiene solo el cuerpo del mensaje y realiza la transformación de los caracteres, en caso de estar habilitada

Métricas

A partir de que el proxy es iniciado, se empiezan a recolectar métricas que luego pueden ser consultadas mediante el administrador. Las métricas son almacenadas en memoria y proveen estadísticas de la sesión actual del proxy.

Las métricas disponibles son:

Accesos

La cantidad de veces que el proxy ha aceptado una conexión

Bytes enviados

La cantidad de bytes que fueron enviados por el proxy

Bytes recibidos

La cantidad de bytes que fueron recibidos por el proxy

Mensajes bloqueados

La cantidad de mensajes que fueron bloqueados

Caracteres convertidos

La cantidad de caracteres que fueron convertidos mediante *leet*.

Protocolo de administrador

Se desarrolló un protocolo que funciona para administrar ciertas características y funcionalidades del proxy. El mismo es del tipo *request-response* y requiere la autenticación con un usuario admin particular para poder modificar o consultar configuraciones.

La función de ayuda es la única que puede ser ejecutada sin estar previamente logueado. Los comandos recibidos son interpretados por línea y luego el proxy otorga una respuesta descriptiva que es precedida por el mensaje de 'ERROR / OK' seguido por una coma.

Las operaciones disponibles son las siguientes:

HELP

Sintaxis	Descripción	Respuesta
HELP\n	Muestra todos los comandos disponibles	OK Welcome to help [comandos disponibles]

LOG

Sintaxis	Descripción	Respuesta
LOG [username] [password]\n	Loguea con dicho usuario y contraseña	OK, Logged in! ERROR, Invalid username! ERROR, invalid pass!

LEET_ON

Sintaxis	Descripción	Respuesta
LEET_ON\n	Habilita el conversor leet	OK, 4pply1ng l33t!

LEET_OFF

Sintaxis	Descripción	Respuesta
LEET_OFF\n	Deshabilita el conversor leet	OK, Leet disabled!

LOG_OUT

Sintaxis	Descripción	Respuesta
LOG_OUT\n	Desloguea el usuario logueado	OK, Good Bye!

BLOCK

Sintaxis	Descripción	Respuesta
BLOCK [username]\n	Bloquea el usuario dado	OK, User blocked!

UNBLOCK

Sintaxis	Descripción	Respuesta
UNBLOCK [username]\n	Desbloquea el usuario dado	OK, User unblocked!

MULTIPLEX

Sintaxis	Descripción	Respuesta
MULTIPLEX [username] [server]\n	Multiplexa el usuario dado al servidor dado	OK, User redirected!

UNPLEX

Sintaxis	Descripción	Respuesta
UNPLEX [username]\n	Quita la multiplexaciones sobre dicho usuario	OK, User unplexed!

PASSCHANGE

Sintaxis	Descripción	Respuesta
PASSCHANGE [username] [new password]\n	Cambia la contraseña por la dada, a dicho usuario	OK, Password changed succesfully!

ACCESSES

Sintaxis	Descripción	Respuesta
ACCESSES\n	Retorna la métrica correspondiente a la cantidad de accesos	OK, [Metrics] Number of total accesses: <%d>

BLOCKED

Sintaxis	Descripción	Respuesta
BLOCKED\n	Retorna la métrica correspondiente a la cantidad de mensajes bloqueados	OK, [Metrics] Number of blocked messages: <%d>

BYTES_SENT

Sintaxis	Descripción	Respuesta
BYTES_SENT\n	Retorna la métrica correspondiente a la cantidad de bytes enviados	OK, [Metrics] Number of bytes sent: <%d>

BYTES_RECEIVED

Sintaxis	Descripción	Respuesta
BYTES_RECEIVED\n	Retorna la métrica correspondiente a la cantidad de bytes recibidos	OK, [Metrics] Number of bytes received: <%d>

CHARACTERS_CONVERTED

Sintaxis	Descripción	Respuesta
CHARACTERS_CONVERTED\n	Retorna la métrica correspondiente a la cantidad de caracteres convertidos	OK, [Metrics] Number of converted characters: <%d>

NOTA: Es importante remarcar que, ante un comando inexistente la respuesta es '*ERROR, Wrong command*', mientras que ante la ejecución de un comando sin estar logueado es *'ERROR, You're not logged in!'*

Registros de acceso

Se decidió utilizar la librería *Logaback* para implementar los logs del proxy. Dichos logs siguen el siguiente patrón (configurado en el *logback.xml*)

```
%date{yy-MMM-dd HH:mm:ss} \t -- \t%-5level\t -- \t%msg%n
```

Cuadro 3: Patrón de los logs

Los logs son guardados en el archivo *proxy-logs.log* y cada uno contiene, como se puede observar en el *Cuadro 3*, la fecha (que incluye hora, minutos y segundos), el tipo de log (error, aviso, información) y el mensaje mismo de cada log.

Problemas Encontrados

Se enumeran a continuación algunos de los problemas encontrados a la hora de desarrollar el proxy

Conversión de caracteres

A la hora de implementar el conversor leet, se encontró un problema con el cacacter '<' ya que al ser enviado sin cuidado es considerado como apertura de un *tag*, ergo el mensaje es considerado como mal formado.

Para solucionar esto, se optó por convertir la c directamente a *<* y que sea codificado después para mostrar correctamente el signo menor.

Otro problema surgió a la hora de convertir algunos pares de caracteres, como por ejemplo 'ce', ya que son convertidos a '<3' una combinación que pidgin interpreta como el emoticón del corazón.

Convivencia de los protocolos

En un principio resultó difícil saber cuando una key tenía que ser interpretada como venida del cliente del proxy xmpp o del admin. Y hasta incluso ha pasado que los handlers se mezclaban.

Para solucionar este problema se decidió crear otra implementación de una conexión, independiente para el administrador.

Por otro lado, si se conecta primero un admin, antes que un cliente al proxy, luego es imposible para el cliente conectarse. Sin embargo todo funciona correctamente, si el orden es inverso.

Aún no hemos podido determinar el porqué de esto.

Identificación de conexión

Al comienzo del desarrollo, resultó difícil encontrar la mejor manera para identificar qué key correspondía a cada cliente, para saber por cual canal había que enviar el mensaje. Esto se solució con lo mencionado en la sección Inicio de la conversación.

Sin embargo, esta solución genera que los mensajes tarden mucho en llegar de un cliente a otro. A nuestro entender esto es causado porque los mensajes entran en una suerte de loop.

Al mismo tiempo, si un usuario escribiera simultáneamente a otros dos usuarios distintos, a uno de ellos le llegaría ambos mensajes.

Seguimos trabajando para encontrar la mejor solución a estos problemas.

Limitaciones de la Aplicación

La principal limitación de la aplicación es que al utilizar una autenticación plana solo se puede conectar a servidores que trabajen con el tag auth. Si el server le responde con un *challenge* al cliente nuestro proxy no podría responder a esto ya que conlleva otro tipo de seguridad que no es soportado.

Por otro lado, cuando más de 5 usuarios se conectan al proxy, las conexiones se vuelven extremadamente lentas, y ya se pierde todo tipo de utilidad al verse perdido el fin principal del proxy.

Posibles Extensiones

Para continuar con el desarrollo y sustentabilidad del proxy, existen varias implementaciones que se pueden realizar. A continuación se enumeran algunos

Métricas

Se pueden implementar mejores métricas, añadiendo más valores y mejores estadísticas. Ejemplo de esto podrían ser

- Métricas sobre las conexiones de los admin.
- Métricas sobre tiempos de conexión.

A su vez, se podría hacer que las métricas no fueran volátiles.

Manejo de errores

Se podría implementar un mejor manejo de los errores y de las excepciones una vez ocurridas.

A su vez, la información que se les da a los usuarios sobre los errores o estados es nula; se podría implementar notificación de errores.

Siguiendo con esa línea de pensamiento, se podría notificar a los usuarios bloqueados que sus mensajes no pueden ser enviados y que están inhabilitados de recibir mensajes, porque se encuentran bloqueados. Aunque se podría pensar que, si alguien quiere tener bloqueado a determinado usuario, puede no querer que dicho usuario lo sepa.

Administrador

Actualmente no hay logs para todo lo que respecta al admin, podrían implementarse en un futuro.

A su vez, se podría aceptar más de un admin, o incluso agregar y quitar de una lista de admins.

Conclusiones

Definitivamente es muy interesante poder entender cómo funciona uno de los protocolos que más utilizamos en nuestra vida diaria, y mucho más aún poder manipularlo y, así, entender un poco mejor cómo funcionan las cosas y el porqué de muchas de las implementaciones que conocemos.

Entender cómo hacer para que el proxy funcione como un pasamanos, que fue el primer paso para un proxy funcional, fue, sin lugar a dudas, el paso más difícil. Una vez entendido eso, y entendido como realizar la arquitectura, el resto fue un poco más sencillo.

Ejemplos de Prueba

Para realizar una prueba de todo lo que es el funcionamiento, se recomiendan los siguientes casos.

Silenciar un usuario

1.- Abrir una conexión netcat **nc localhost 5224**

Reemplazar localhost por el host donde este corriendo el proxy

2.- Loguearse ingresando el comando **login sebas-admin@protos-tpe**

3.- Ingresar el comando **BLOCK usuario@servidor**

Reemplazar *usuario@servidor* por el usuario que se quiere silenciar

Escribir un mensaje desde un usuario silenciado

1.- Iniciar sesión con un usuario que esté silenciado

2.- Escribir un mensaje a otro usuario

Verificar los logs

1.- Dentro del root del proyecto se encuentra un archivo llamado proxy-logs.log donde estan los logs del proyecto

2.- Hacer un tail -f del archivo para verlo actualizado

Aplicar Leet

1.- Abrir una conexión netcat **nc localhost 5224**

Reemplazar localhost por el host donde este corriendo el proxy

2.- Loguearse ingresando el comando **login sebas-admin@protos-tpe**

3.- Aplicar el conversor ingresando **LEET_ON**

Enviar mensajes con transformaciones

1.- Iniciar sesión con un usuario.

2.- Escribir un mensaje a otro usuario

Métricas

1.- Abrir una conexión netcat **nc localhost 5224**

Reemplazar localhost por el host donde este corriendo el proxy

2.- Loguearse ingresando el comando **login sebas-admin@protos-tpe**

3.- Solicitar las métricas ingresando **ACCESSES / BLOCKED / BYTES_READ ...**

Guía de Instalación y Configuración

Test Plan

- Descargar Prosody (Server)
- Descargar Pidgin (Client)
- Copiar el archivo test.cfg.lua en la carpeta /etc/prosody/conf.avail y crear la imagen en/etc/prosody/conf.d
- Crear como host (en /etc/hosts) protos-tpe y direccionarlo a 127.0.0.1

Run

- Crear un usuario en el servidor
    - Se puede hacer desde pidgin:
        - Conectar un XMPP nuevo
        - Inventar username
        - El dominio tiene que ser el lugar donde este guardado el server (protos-tpe)
        - Poner en true el campo para crear en el server
        - En advanced utlizar el puerto del server (5228) y poner usar encriptacion si esta disponible
        - Conectar - Va a pedir registrar, usar los mismos campos (importante agregar el @protos-tpe al usuario) y recordar contraseña
        - SE CREO EL USUARIO * Desconectar el cambiar el puerto por el puerto del proxy (5225)
- Darle play al proxy (desde java)
- Conectar el usuario al proxy y un usuario directamente al server
- Agregar un buddy
- Empezar a hablar

Ejemplos de Configuración y Monitoreo

A continuación se muestran algunos ejemplos del uso del protocolo de administración, junto con algunos de los comandos disponibles.

```
sebastian@Sebastians:~$ nc 127.0.0.1 5224
LOG sebas-admin@protos-tpe 123456789
OK, Logged in!
LEET_ON
OK, 4pply1ng l33t!
LOG_OUT
OK, Good Bye!
sebastian@Sebastians:~$ nc 127.0.0.1 5224
LOG sebas-admin@protos-tpe 123456789
OK, Logged in!
LEET_ON
OK, 4pply1ng l33t!
sebastian@Sebastians:~$ ^C
sebastian@Sebastians:~$ nc 127.0.0.1 5224
LOG sebas-admin@protos-tpe 123456789
OK, Logged in!
LEET_ON
OK, 4pply1ng l33t!
LEET_OFF
OK, Leet disabled!
BLOCK gilada@protos-tpe
OK, User blocked!
BLOCKED
OK, [Metrics] Number of blocked messages: 0
```

Cuadro 4: Ejemplos de uso de configuraciones y monitoreo

Documento de Diseño del Proyecto

Si recordamos la *Imagen 1* será mucho más fácil entender cómo funciona el diseño del proyecto.

Cada cliente está conectado al proxy por un *socket* channel distinto, dependiendo del cliente. Luego el proxy crea un *socket channel* hacia el servidor. Cada uno de estos también es independiente y único para cliente.

Al intentar un nuevo cliente conectarse al proxy, este crea un nuevo *socket channel* hacia el cliente, y otro hacia el servidor, previa aceptación de la conexión.

La flecha verde simboliza el flujo entero de una conexión.

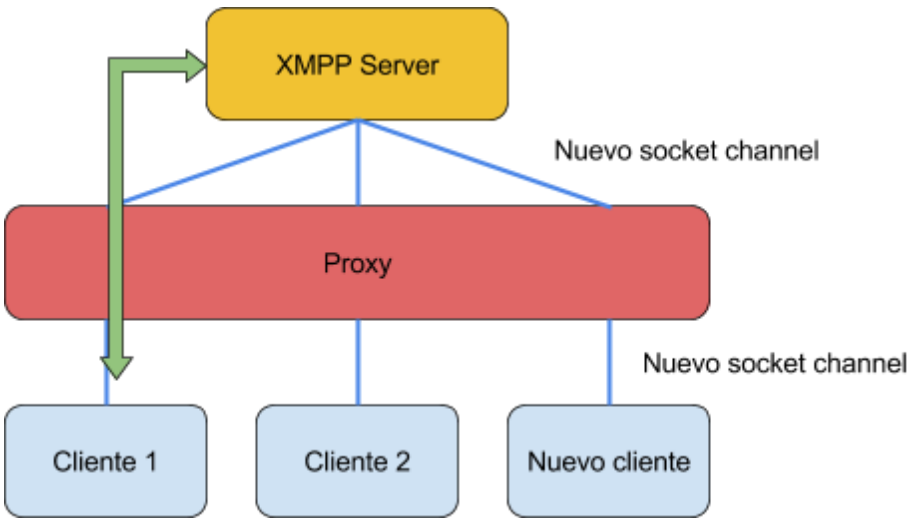

Imagen 2: Representación de una conexión entre el cliente y el servidor

Bibliografía

http://logback.qos.ch/manual/layouts.html

https://examples.javacodegeeks.com/core-java/nio/java-nio-socket-example/

http://stackoverflow.com/questions/6438345/how-to-create-a-jabber-xmpp-proxy-logging-service

http://stackoverflow.com/questions/2031163/when-to-use-the-different-log-levels

ftp.rfc-editor.org/in-notes/std/std68.txt