Pre-entrega Trabajo Práctico Especial

Protocolos de Comunicación

Andrés Mata Suarez - 50143
Jimena Pose - 49015
Pablo Ballesty - 49359

2011 - Segundo cuatrimestre

Índice

1. Documentación relevante para el desarrollo

Para el análsis del trabajo práctico se utilizan los siguientes documentos

- RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core
- RFC 6121: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence
- RFC 6122: Extensible Messaging and Presence Protocol (XMPP): Address Format

2. Protocolos a desarrollar

A continuación se presenta el RFC del protocolo *configurotocol 1.0*, diseñado para manejar la configuración del servidor proxy.

```
                              Configurotocol 1.0


Resumen
Configurotocol es un protocolo desarrollado para poder configurar en tiempo real
y de forma remota la aplicación proxy Isecu.

Estado del documento
Este protocolo forma parte de la entrega del Trabajo Práctico Especial para la
materia de Protocolos de Comunicación, de la carrera Ingeniería Informática del
Instituto Teconológico de Buenos Aires (ITBA).


1. Introducción

Configurotocol es un protocolo sin estado que permite a una entidad cliente con-
sultar y setear parámetros de configuración de la aplicación proxy Isecu.

2. Descripción

El protocolo está diseñado para poder ejecutar comandos de configuración en tiem-
po real y de forma remota. Un comando va a estar formado por instrucciones. A
continuación se definen ambos conceptos y sus respectivos formatos.

2.1 Objeto

Un objeto es un conjunto de 2 elementos: clave y valor. Un objeto es igual a otro
si ambos contienen la misma clave. La clave es case insensitive y el valor es
key sensitive.
Un objeto puede ser:

                      simple -->   clave : "valor"

                                        ó

          compuesto --> clave : ["valor1", "valor2", ... , "valorn"]

2.2 Comando

Un comando es un conjunto de objetos dentro de llaves. Los objetos están separados
por comas. Que un comando sea un conjunto, implica que no contiene elementos
repetidos y que el orden en que se encuentran es irrelevante.
Ejemplo de comando:

                {nombre : "Thulsa", apellido : "Doom"}
```

3. Comandos válidos

Anteriormente se definió el formato de los comandos; en esta sección se especifican los comandos que soporta el protocolo.
Un comando es considerado válido si cumple con las siguientes reglas:

A. Respeta el formato de comando especificado en 2.2.
B. Posee al menos tres objetos, "user", "password" y "type", siendo todos ellos de formato simple.
C. El objeto "type" debe tener como valor "query" o "assignation".
D. En caso de ser del tipo "query" debe cumplir con lo especificado en 3.1.
E. En caso de ser del tipo "assignation" debe cumplir con lo especificado en 3.2.

3.1 Comandos de tipo "query"

Un comando que cumple con las reglas A, B, C y es del tipo "query" se considera válido si además posee un objeto con clave "parameter", cuyo valor sea alguno de los siguientes:

- server
- blacklist
- caccess
- multiplex
- silence
- filter

3.2 Comandos de tipo "assignation"

Un comando que cumple con las reglas A, B, C y es del tipo "assignation" se considera válido si posee al menos uno de los siguientes objetos, y son todos válidos:

3.2.1 Server
El objeto server debe ser de formato compuesto, y debe tener exactamente dos valores. El primero debe ser la dirección del servidor y el segundo el puerto.
Ejemplo:
```
{ user:"foo", password:"foo", type:"assignation",
server:["xmpp.example.com","5222"] }
```

3.2.2 Blacklist
El objeto blacklist debe ser de formato compuesto y de algún tipo de los siguientes:

Tipo "range":
```
 blacklist:["range","user","from","to"]
```

Tipo "logins":
```
 blacklist:["logins","user","qty"]
```

Tipo "net":
```
 blacklist:["net","ip"]
 blacklist:["net","network","netmask"]
```

3.2.3 Acceso concurrente
El objeto caccess debe ser de formato compuesto y debe respetar el siguiente formato:
```
 caccess:["user","qty"]
```

3.2.4 Multiplexador de cuentas
El objeto multiplex debe ser de formato compuesto y debe respetar el siguiente formato:
```
  multiplex:["jid", "serverto"]
```

```
3.2.5 Silenciar usuarios
El objeto silence debe ser de formato simple y debe respetar el siguiente forma-
to:
 silence:"user"

3.2.6 Filtros
El objeto filter debe ser de formato compuesto y debe respetar el siguiente for-
mato:
 filter["filtername","on/off"]

4. Respuestas
Luego de la ejecución de un comando la aplicación enviará una respuesta.
Toda respuesta va a contener el objeto status, con valor "OK" o "ERROR" corres-
pondiente a una respuesta satisfactoria o no, respectivamente. Luego de cualquier
comando de tipo "assignation" se va a responder con una respuesta conformada
sólo con un objeto status. En caso de que el comando haya sido de tipo "query"
los resultados de esa query se enviarán en formato compuesto o simple, depen-
diendo de la cantidad de resultados obtenidos, en el objeto "data".
```

3. Potenciales problemas y dificultades

3.1. Comunicación encriptada

Como se especifica en el RFC 6120, cuando dos entidades quieren comunicarse, negocian qué método de encriptación utilizarán, el server manda sus *features* y el cliente avisa cual utilizará. Luego comienzan nuevamente la sesión, utilizando el método escogido.

Para el servidor proxy, esto va a resultar un problema, ya que luego de que ambas entidades escojan un protocolo de comunicación encriptado (los dos que se mencionan son TLS y SASL) los paquetes comenzarán a llegar encriptados y por ende, no podremos realizar algunos de los requerimientos que se piden (e.g filtro de l33t).

Para evitar que pasen por el proxy paquetes encriptados, sin que se puedan decodificar, se decide que al momento en que las dos entidades hayan escogido el método a utilizar, sea el server proxy el que utilice ese método para ambos lados de la conexión. De esta manera, sabrá encriptar y desencriptar los mensajes de cada parte de la conexión. Por ejemplo, si quiere reenviar todo lo recibido del cliente al servidor, antes va a tener que desencriptarlo y luego encriptarlo para enviarlo al servidor.

3.2. Transparencia

El proxy XMPP a desarrollar debería actuar de manera completamente invisible frente a cada par de entidades (cliente y servidor) conectadas. En otras palabras, el flujo de información que envía una parte debería arribar sin ningún tipo de alteración al otro extremo de la conexión, siempre y cuando no estén activas ninguna de las funciones de transformación ofrecidas por la aplicación. Por su parte, el resto de los requerimientos funcionales no deberían modificar en lo más mínimo el *stream* XML enviado entre pares.

3.3. Concurrencia

Frente a cantidades superlativas de conexiones, el proxy debería estar diseñado para trabajar de manera performante, optimizando el manejo de hilos de ejecución. Para tal fin, se opta por la implementación de un patrón de *pool* de *threads*. La creación atolondrada de *threads* podría consumir demasiados recursos en un período relativamente corto de tiempo; por el contrario, la constante reutilización de hilos de ejecución para nuevas conexiones podría provocar un verdadero cuello de botella en la performance y el tiempo de ejecución de la aplicación. El objetivo de este inciso radica en encontrar el balance correcto entre la creación y reutilización de threads, de tal manera que el sistema funcione en condiciones óptimas.

Por supuesto, la destrucción criteriosa de hilos inactivos es un factor tan importante como sendos anteriores. De más está decir que el consumo de recursos del sistema debería ser el mínimo posible.

3.4. Manejo de grandes *streams* de información

Los *streams* de información que se consideren de gran tamaño deben ser almacenados temporalmente en disco. Obviar esta política implicaría riesgo de agotación de la memoria y, en consecuencia, la pérdida de la estabilidad deseada en la aplicación. Debido a esto, es necesario implementar un módulo eficiente con la función de mantener

ciertos *streams* en disco e ir llevándolos a memoria a medida que sea necesario. La aplicación no debería dejar de funcionar (o comenzar a funcionar erráticamente) debido al tamaño de los *streams* que se manejen.

De nuevo, se espera que el manejo de recursos sea eficiente: no es deseable la creación excesiva de archivos en disco así como tampoco lo es la no eliminación de archivos que ya no están en uso por la aplicación.

3.5. Controles y aplicaciones externas

En caso de que algún control efectuado por la aplicación de proxy (por ej.: control de accesos) tenga efecto sobre cierta entidad en una conexión, el mensaje de error devuelto deberá tener la forma de un mensaje de error XMPP. Una aplicación externa que utilize el proxy no debería tener motivos para sospechar que un error del tipo mencionado no proviene del otro *endpoint* de conexión.

4. Ambiente de desarrollo y testing

Como servidor de XMPP se utilizará **ejabberd** (`http://www.ejabberd.im/`), el cual puede instalarse en Ubuntu directamente desde los repositorios ejecutando desde consola:

```
sudo apt-get install ejabberd
```

Como cliente se utilizará **Pidgin** (`http://www.pidgin.im/`), el cual peude instalarse en Ubuntu directamente de los respositorios ejecutando desde consola:

```
sudo apt-get install pidgin
```

5. Casos de prueba a realizar

Para cada caso de testeo, se consideran los siguientes usuarios:

- **cthulhu**, contraseña **fhtagn**.
  Usuario con permisos administrativos. Cada vez que se pide utilizar el protocolo *configurotocol* para realizar algún tipo de configuración, se asume que se está utilizando a este usuario.
- **dagon**, contraseña **dagon**.
- **shubniggurath**, contraseña **shubniggurath**.

5.1. XXXXXXXXXXX CASOS DE TESTEO QUE FALTAN XXXXXXXXXXXXXX

!!!!!ANOTO LOS QUE NO HICE, NO SE SI HAY QUE HACER PARA TODOS IGUALMENTE !!!!!!!!!!!!!
!!!!! EN LOS CASOS DE TESTEO CORRESPONDIENTES A BLACKLIST, FALTAN METER VALORES NUMERICOS PARA LAS IP !!!!!!!!!

- Modo de uso
- concurrencia
- encadenamiento de proxis
- logging
- multiplexador de cuentas
- transofrmaciones

5.2. Controles

Funcionalidad	Por rango de horarios
Tipo de test	Positivo
Descripción	Utilizar el protocolo *configurotocol* para exigir que el usuario **dagon** sólo pueda acceder de 6:00 a 18:00 horas. Iniciar sesión con dicho usuario en dicho rango horario.
Resultado esperado	El usuario **dagon** inicia sesión sin problemas.

Funcionalidad	Por rango de horarios
Tipo de test	Negativo
Descripción	Utilizar el protocolo *configurotocol* para exigir que el usuario **dagon** sólo pueda acceder de 6:00 a 18:00 horas. Iniciar sesión con dicho usuario fuera de ese rango horario.
Resultado esperado	El usuario **dagon** no tiene permitido el inicio de sesión en dicho rango horario. Se devuelve mensaje de error acorde.

Funcionalidad	Por cantidad de logins exitosos por usuario y día
Tipo de test	Positivo
Descripción	Asegurarse de que el usuario **dagon** no haya iniciado sesión anteriormente en el día. Utilizar el protocolo *configurotocol* para exigir que el usuario **dagon** sólo pueda acceder al sistema un máximo de 1 (una) vez por día. Iniciar sesión en el sistema como dicho usuario.
Resultado esperado	El usuario **dagon** inicia sesión sin problemas.

Funcionalidad	Por cantidad de logins exitosos por usuario y día
Tipo de test	Negativo
Descripción	Asegurarse de que el usuario **dagon** no haya iniciado sesión anteriormente en el día. Utilizar el protocolo *configurotocol* para exigir que el usuario **dagon** sólo pueda acceder al sistema un máximo de 1 (una) vez por día. Iniciar sesión en el sistema como dicho usuario. Cerrar sesión. Iniciar sesión una vez más.
Resultado esperado	El usuario **dagon** ya cumplió su cuota diaria de accesos. El segundo login no es aceptado. Se devuelve mensaje de error acorde.

Funcionalidad	Por lista negra (dirección IP)
Tipo de test	Positivo
Descripción	Utilizar el protocolo *configurotocol* para impedir conexiones entrantes de la dirección XXX.XXX.XXX.XXX. Iniciar sesión como **dagon** desde la dirección YYY.YYY.YYY.YYY.
Resultado esperado	El usuario **dagon** inicia sesión sin problemas.

Funcionalidad	Por lista negra (dirección IP)
Tipo de test	Negativo
Descripción	Utilizar el protocolo *configurotocol* para impedir conexiones entrantes de la dirección XXX.XXX.XXX.XXX. Iniciar sesión como **dagon** desde la dirección XXX.XXX.XXX.XXX.
Resultado esperado	La dirección XXX.XXX.XXX.XXX se encuentra en la lista negra. No se permite el inicio de sesión. Se devuelve mensaje de error acorde.

Funcionalidad	Por lista negra (redes IP)
Tipo de test	Positivo
Descripción	Utilizar el protocolo *configurotocol* para impedir conexiones entrantes del rango de direcciones XXX.XXX.XXX.XXX/ZZ. Iniciar sesión como **dagon** desde la dirección YYY.YYY.YYY.YYY. Iniciar sesión como **shubniggurath** desde la dirección VVV.VVV.VVV.VVV.
Resultado esperado	Ambos usuarios inician sesión sin problemas.

Funcionalidad	Por lista negra (redes IP)
Tipo de test	Negativo
Descripción	Utilizar el protocolo *configurotocol* para impedir conexiones entrantes del rango de direcciones XXX.XXX.XXX.XXX/ZZ. Iniciar sesión como **dagon** desde la dirección YYY.YYY.YYY.YYY. Iniciar sesión como **shubniggurath** desde la dirección VVV.VVV.VVV.VVV.
Resultado esperado	No se permite ninguno de los dos inicios de sesión. Se devuelven mensajes acordes para cada instancia de la aplicación.

Funcionalidad	Por cantidad de sesiones concurrentes
Tipo de test	Positivo
Descripción	Utilizar el protocolo *configurotocol* para restringir la cantidad máxima de sesiones concurrentes del usuario **dagon** a 3. Iniciar sesión como dicho usuario desde 2 instancias de la aplicación distintas.
Resultado esperado	Se permite el inicio de sesión en cada instancia del programa.

Funcionalidad	Por cantidad de sesiones concurrentes
Tipo de test	Negativo
Descripción	Utilizar el protocolo *configurotocol* para restringir la cantidad máxima de sesiones concurrentes del usuario **dagon** a 3. Iniciar sesión como dicho usuario desde 3 instancias de la aplicación distintas. Ejecutar una nueva instancia e iniciar sesión como el mismo usuario una vez más.
Resultado esperado	Se permite el inicio de sesión en todas las instancias excepto en la última. Se devuelve mensaje de error acorde en esta última.

Funcionalidad	Por silenciamiento de usuarios
Tipo de test	Positivo
Descripción	Utilizar el protocolo *configurotocol* para asegurarse de que el usuario **dagon** no se encuentre silenciado. Iniciar sesión como dicho usuario y empezar a ejecutar comandos XMPP.
Resultado esperado	El usuario **dagon** puede comunicarse sin problemas.

Funcionalidad	Por silenciamiento de usuarios
Tipo de test	Negativo
Descripción	Utilizar el protocolo *configurotocol* para silenciar al usuario **dagon**. Iniciar sesión como dicho usuario y empezar a ejecutar comandos XMPP.
Resultado esperado	El usuario **dagon** se encuentra silenciado. Se devuelve mensaje de error acorde.