Trabajo Práctico Especial

Protocolos de Comunicación

Revisión correspondiente a la entrega:

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

2011 - Segundo cuatrimestre

Índice

1. Descripción detallada del protocolo desarrollado

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.
Se basa en el formato JSON (http://www.json.org).

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. Tanto la clave como el valor son case
sensitive.
Un objeto puede ser:

                    simple -->   "clave" : "valor"

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

O puede contener otro objeto:

        {"calve":"valor","claveObjeto":{"clave1":"valor2","clave2":"valor2"}}

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 dos objetos, "auth" (Especificado en la sección 3.1) y "type" (Especificado en la sección 3.2).
C. En caso de ser del tipo "query" debe cumplir con lo especificado en 3.3.
D. En caso de ser del tipo "assignation" debe cumplir con lo especificado en 3.4.
E. En caso de ser del tipo "delete" debe cumplir con lo especificado en 3.5.

3.1 Objeto "auth"

El objeto auth es obligatorio, debe tener el siguiente formato:

```
"auth" : ["user", "pass" ]
```

Donde "user" corresponde al nombre de usuario del administrador y pass a su contraseña.

3.2 Objeto "type"

El objeto type es obligatorio, debe ser de tipo simple y contener alguno de los siguientes valores:

- "query"
- "assignation"
- "delete"

3.2.1 Comandos de tipo "query"

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

- caccess
- multiplex
- silence
- leet
- hash
- rangeBlacklist
- loginsBlacklist
- ipBlacklist
- netBlacklist

3.2.2 Comandos de tipo "assignation"

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

3.2.2.1 Blacklist

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

- Tipo "range":
  "blacklist":["range","jid","10:15:00","18:30:00"]

- Tipo "logins":
  "blacklist":["logins","jid","5"]

- Tipo "ip":
  "blacklist":["ip","10.0.0.3"]

- Tipo "net":
  "blacklist":["net", "10.0.0.0/24"}

3.2.2.2 Acceso concurrente

El objeto caccess debe ser de formato compuesto y debe respetar el siguiente formato:

```
"caccess":["jid", "3"]
```

3.2.2.3 Multiplexador de cuentas

El objeto multiplex debe ser de formato compuesto y debe respetar el siguiente formato:

```
"multiplex":["jid", "10.0.0.1"]
```

3.2.2.4 Silenciar usuarios

El objeto silence debe ser de formato simple y debe respetar el siguiente formato:

```
"silence":"jid"
```

3.2.2.5 Filtros

El objeto filter debe ser de formato compuesto y debe respetar el siguiente formato:

```
"filter":["filtername", "jid", "state"]
```

Donde filtername puede ser leet o hash y state puede ser on u off.

3.2.3 Comandos de tipo "delete"

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

3.2.3.1 Blacklist

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

- Tipo "range":
  `"blacklist":["range","jid"]`
- Tipo "logins":
  `"blacklist":["logins","jid"]`
- Tipo "ip":
  `"blacklist":["ip","10.0.0.3"]`
- Tipo "net":
  `"blacklist":["net", "10.0.0.0/24"}`

3.2.3.2 Acceso concurrente

El objeto caccess debe ser de formato simple y debe respetar el siguiente formato:

```
"caccess":"jid"
```

3.2.3.3 Multiplexador de cuentas

El objeto multiplex debe ser de formato simple y debe respetar el siguiente formato:

```
"multiplex":"jid"
```

3.2.3.4 Silenciar usuarios

```
El objeto silence debe ser de formato simple y debe respetar el siguiente forma-
to:
   "silence":"jid"

3.2.3.5 Filtros

El objeto filter debe ser de formato compuesto y debe respetar el siguiente for-
mato:
   "filter":["filtername", "jid"]
Donde filtername puede ser leet o hash y state puede ser on u 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" o "delete" 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, dependiendo de la cantidad de resultados obtenidos, en el objeto "data".
```

2. Problemas encontrados durante el diseño y la implementación

3. Limitaciones de la aplicación

4. Posibles extensiones

5. Conclusiones

6. Ejemplos de testeo

Para la ejecución de los casos de prueba, se va a contar con los siguientes usuarios:

- **foo** con contraseña **123123123**
- **bar** con contraseña **123123123**
- **baz** con contraseña **123123123**
- **foobar** con contraseña **123123123**

6.1. Testeos de requerimientos funcionales

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

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

Funcionalidad	Por rango de horarios
Tipo de test	Negativo
Descripción	Utilizar el protocolo *configurotocol* para exigir que el usuario **foo** sólo pueda acceder de 19:00 a 06:00 horas. Iniciar sesión con dicho usuario dentro de ese rango horario.
Resultado esperado	El usuario **foo** no tiene permitido el inicio de sesión en dicho rango horario. Se recibe un error acorde al protocolo. Que el configurotocol permita este seteo.

Funcionalidad	Por cantidad de logins exitosos por usuario y día
Tipo de test	Positivo
Descripción	Asegurarse de que el usuario **bar** no haya iniciado sesión anteriormente en el día. Utilizar el protocolo *configurotocol* para exigir que el usuario **bar** 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 **bar** inicia sesión sin problemas.

Funcionalidad	Por cantidad de logins exitosos por usuario y día
Tipo de test	Negativo
Descripción	Asegurarse de que el usuario **bar** no haya iniciado sesión anteriormente en el día. Utilizar el protocolo *configurotocol* para exigir que el usuario **bar** 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 **bar** ya cumplió su cuota diaria de accesos. El segundo login no es aceptado. Se devuelve mensaje de error acorde al protocolo.

Funcionalidad	Por lista negra (dirección IP)
Tipo de test	Positivo
Descripción	Utilizar el protocolo *configurotocol* para impedir conexiones entrantes de la dirección 192.168.1.50. Iniciar sesión como **baz** desde la dirección 192.168.1.51.
Resultado esperado	El usuario **baz** 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 192.168.1.50. Iniciar sesión como **baz** desde la dirección 192.168.1.50.
Resultado esperado	La dirección 192.168.1.50 se encuentra en la lista negra. No se permite el inicio de sesión. Se devuelve mensaje de error acorde al protocolo.

Funcionalidad	Por lista negra (redes IP)
Tipo de test	Positivo
Descripción	Utilizar el protocolo *configurotocol* para impedir conexiones entrantes del rango de direcciones 192.168.1.0/25. Iniciar sesión como **foobar** desde la dirección 192.168.1.130. Iniciar sesión como **baz** desde la dirección 192.168.1.254.
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 192.168.1.0/25. Iniciar sesión como **foobar** desde la dirección 192.168.1.126. Iniciar sesión como **bar** desde la dirección 192.168.1.10.
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 **baz** a 3. Iniciar sesión como dicho usuario desde 2 clientes distintos.
Resultado esperado	Se permite el inicio de sesión en cada instancia del programa.

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 **baz** a 3. Iniciar sesión como dicho usuario desde 3 clientes distintos. Utilizar un nuevo cliente e iniciar sesión nuevamente.
Resultado esperado	Se permite el inicio de sesión en el último cliente, y el primer cliente que se utilizó se desconecta recibiendo un error acorde al protocolo.

Funcionalidad	Por silenciamiento de usuarios
Tipo de test	Positivo
Descripción	Utilizar el protocolo *configurotocol* para asegurarse de que el usuario **foobar** no se encuentre silenciado. Iniciar sesión como dicho usuario y emitir mensajes hacia el usuario **foo** que estará conectado. Luego el usuario **foo** emite mensajes hacia **foobar**.
Resultado esperado	Los usuarios se comunican sin problemas.

Funcionalidad	Por silenciamiento de usuarios
Tipo de test	Negativo
Descripción	Utilizar el protocolo *configurotocol* para silenciar al usuario **foobar**. Iniciar sesión como dicho usuario y emitir mensajes hacia el usuario **foo** que estará conectado. Luego **foo** envía mensajes hacia **foobar** y hacia **bar**.
Resultado esperado	**foo** y **foobar** no pueden comunicarse, y **bar** recibe los mensajes de **foo**.

Funcionalidad	Filtros(L33t)
Tipo de test	Positivo
Descripción	Mediante *configurotocol* activar el filtro de **L33t**. Emitir mensajes desde **foo** hacia **bar** que posean vocales.
Resultado esperado	**bar** recibe los mensajes en formato **L33t**.

Funcionalidad	Filtros(L33t)
Tipo de test	Positivo
Descripción	Mediante *configurotocol* activar el filtro de **L33t**. Emitir mensajes desde **foo** hacia **bar** que posean vocales.
Resultado esperado	**bar** recibe los mensajes en formato **L33t**.

Funcionalidad	Filtros(Hash)
Tipo de test	Positivo
Descripción	Mediante *configurotocol* activar el filtro de **Hash**. Enviar un archivo desde **foo** hacia **bar**. Asegurarse que se envía el hash correspondiente.
Resultado esperado	**bar** recibe el archivo sin problemas, con el hash generado.

Funcionalidad	Filtros(Hash)
Tipo de test	Positivo
Descripción	Mediante *configurotocol* activar el filtro de **Hash**. Enviar un archivo desde **foo** hacia **bar**. Quitar el hash del mensaje.
Resultado esperado	**bar** recibe el archivo sin problemas, con el hash generado.

Funcionalidad	Filtros(Hash)
Tipo de test	Negativo
Descripción	Mediante *configurotocol* activar el filtro de **Hash**. Enviar un archivo desde **foo** hacia **bar**. Alterar el hash.
Resultado esperado	**bar** no recibe el archivo.

6.2. Testeos de requerimientos no funcionales

Requerimiento no funcional	Envío de archivos
Tipo de test	Positivo
Descripción	Desde el usuario **foo** enviar un archivo a **bar** de un tamaño de 30Mb.
Resultado esperado	**bar** recibe el archivo sin problemas.

Requerimiento no funcional	Concurrencia
Tipo de test	Positivo
Descripción	Configurar JMeter para que realice n conexiones al servidor proxy y emita mensajes. Ir aumentando n.
Resultado esperado	Para n aceptable (analizaremos durante el desarrollo cuanto sería un n aceptable) no se rechazan conexiones y pueden emitirse los mensajes.

7. Guía de instalación detallada y precisa

8. Instrucciones para la configuración

9. Ejemplos de configuración

10. Documento de diseño del proyecto