

Control remoto con un *bot* Jabber

# Contacto Interior

Para acceder a una LAN traspasando un cortafuegos, se necesita una puerta trasera secreta o un agente colaborador dentro. Un cliente Jabber dentro de la LAN conecta con un servidor Jabber público y esperará que aparezcan instrucciones en forma de mensajes instantáneos enviados por su conocido de Internet. **POR MICHAEL SCHILLI**

Por supuesto, que una manera de realizar tareas, desde Internet, en una red local es abrir un agujero a través del cortafuegos y conectar con un servidor web local. Servicios como <http://www.dyn-dns.org/> (entre otros) permiten un acceso casi estático a través de la dirección IP dinámica que asigna el proveedor de Internet.

Un agente o “bot” (palabra derivada de “robot”) hace la vida más fácil: Un cliente de mensajería detrás del cortafuegos puede conectar con una red pública de mensajería Jabber y aceptar ordenes en formato de mensajes de texto. El cliente descrito en este artículo solo aceptará ordenes desde clientes de su lista de conocidos y solo permitirá cuatro acciones: cargar la comprobación para el ordenador *bot*, consultar la dirección pública del enrutador (orden: *ip*) y encender y apagar las luces de mi apartamento en San Francisco (*lamp on|lamp off*).

El guión *agent.pl* (Véase el Listado 1) necesita *Log::Log4perl* y transacciones de “log” en un archivo llamado

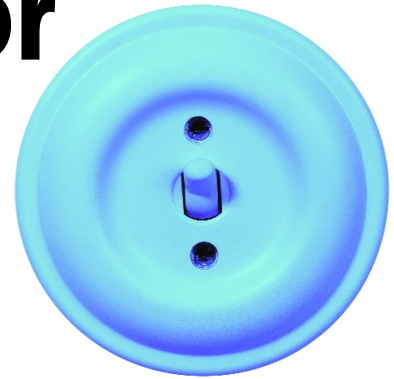
*/tmp/agent.log*. La línea 33 de este guión crea un nuevo objeto *Net::Jabber::Client* que implementa el cliente de mensajería instantánea que actuará como agente.

Antes *agent.pl* entra en el bucle principal de eventos en la línea 83, se necesita definir unas cuantas retro-llamadas para varios eventos en la línea 35ff. El manejador *onauth* en la línea 68 se llamará después de que el cliente haya ingresado en el servidor usando las credenciales especificadas en la línea 23. El manejador entonces llamará a *RosterGet()* para que vaya a buscar la lista de conocidos y la almacene en una variable global *%ROSTER*.

El método siguiente, *Presence()*, envía un mensaje *presence* a todos los clientes en la lista para decirles que *agent.pl* está en línea. A partir de este punto, un cliente *gaim* que haya ingresado como el usuario *mikes-agent-sender* mostrará a *mikes-agent-receiver* como cliente activo en su propia lista de conocidos (Véase la Figura 2).

La retro-llamada *message* en la línea 37 se invocará cuando alguien envíe un mensaje a *agents.pl*. Cualquier cliente en la red Jabber puede hacer esto y es por eso que la línea 45 comprueba si el emisor es un amigo. En este caso *mikes-agent-sender* es el único al que se le permite enviar un mensaje, ya que es la única entrada en la lista de conocidos del cliente (Véase la sección “Instalación”). La función simplemente descarta todas las demás peticiones, almacena un mensaje de información en archivo de “log” y vuelve al bucle principal en la línea 50.

El método *getBody()* en la línea 57 del guión extrae la orden de control enviada



con el mensaje de texto y la pasa a la función *run\_cmd* definida en la línea 93.

En la línea 83, *Execute()* conecta con el servidor de Jabber en *jabber.org* e ingresa como *mikes-agent-receiver*. Si se pierde temporalmente la conexión el bucle principal, que se ejecuta indefinidamente, la rescatará. Si también se detiene porque se han producido muchos errores, la línea 90 los eliminará y parará el programa.

Para estar seguro de que el agente empiece a funcionar cuando se inicialice el sistema, se añade lo siguiente:

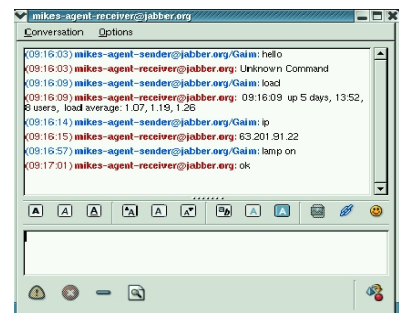


Figura 1: El “bot” detrás del cortafuegos, ejecuta las ordenes que se le envían mediante un cliente Jabber basado en Internet.



Figura 2: El “bot” aparece ahora en la lista de conocidos del emisor.



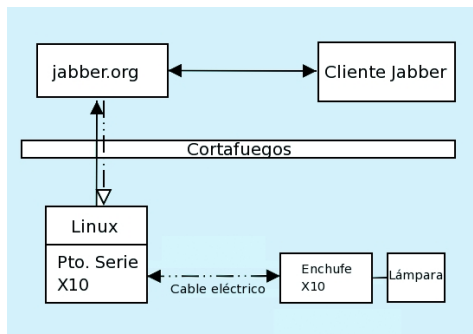


Figura 3: Visión general del interruptor de luz controlado por un "bot".

```
x:3:respawn:su Z
username -cZ
/usr/bin/agent
```

a `/etc/inittab`. Este código también se cerciora de que el agente consiga reiniciarse inmediatamente si deja de funcionar por cualquier razón.

## Arriba y Abajo

El agente descubre la dirección externa del enrutador enviando una solicitud web a la URL pública `http://perlmeister.com/cgi/whatsmyip`. El objetivo es un simple guión que devuelve la dirección del cliente solicitante:

```
print Z
"Content-Type: text/html\n\n";
print $ENV{REMOTE_ADDR}, "\n";
```



Figura 4: La dispositivo X10 espera a la señal y conecta y desconecta la lámpara.

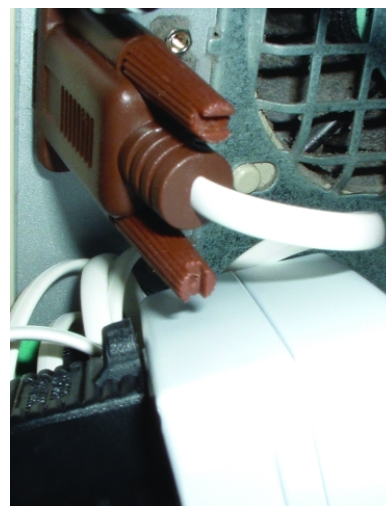


Figura 5: La unidad de control X10 con un conector serie puede enviar señales desde el ordenador a los dispositivos X10 a través de la instalación eléctrica.



Figura 6: Encendiendo la lámpara del dormitorio vía Internet.

Para hacer esto `agent.pl` usa `LWP::Simple`; si el mensaje de texto que el cliente Jabber recibe es `IP`, la función `get` de la línea 99 toma el contenido de la página web.

Determinar la carga actual del sistema sigue un patrón similar: La línea 17 llama a `uptime` y el resultado pasa de vuelta a la línea 55. La siguiente llamada a `chomp` elimina el contenido sobrante y la línea 62 amontona el resultado en el cuerpo del mensaje;

el método `send` envía este resultado hacia el compañero solicitante.

¿Pero como hace un agente que está funcionando en un ordenador Linux, para encender la luz del dormitorio?

La Figura 3 muestra la configuración. En EE.UU. se utiliza mucho una tecnología conocida como X10 para transmitir señales a través del cableado de la instalación eléctrica de los hogares (por supuesto que los equipos americanos solo funcionan con el voltaje de ese país, aunque en Europa también se venden productos que utilizan esta tecnología adaptados a los valores europeos [5]). Estos equipos se comunican con el ordenador a través de interfaces serie o USB. Hay disponible una amplia gama de dispositivos X10 para diversas tareas de automatización doméstica. Además de

simples interruptores, se pueden encontrar sistemas con capacidades X10 en cámaras de vigilancia, sensores de movimiento, sistemas de alarma, reproductores mp3, televisiones y detectores de metales.

Cada unidad de control X10 (Figura 4) tiene un código de casa (A-P) y un código de unidad (1-16) que la unidad de control (por ejemplo la de la Figura 5) ha de seleccionar para encender la lámpara correcta (en vez de la del vecino). X10 no es caro (en EE.UU.): Un kit básico con cuatro componentes con toda clase de extras y un control remoto cuesta en algunos sitios entre 50\$ y 100\$ en [3].

El listado `lamp.pl` muestra un guión corto que envía los códigos a través del puerto serie al controlador de la lámpara. El guión solo usa `Device::ParallelPort` y `ControlX10::CM11` de CPAN, para direccionar la unidad usando el código de casa/unidad en la línea 38. El subsiguiente `send()` con el código y una "J" (para encender) o una "K" (para apagar) activa el dispositivo indicado. El puerto serie usado en este ejemplo en la línea 34 es `/dev/ttyS0`, donde la pequeña caja blanca que se ve en la Figura 6 está conectada al primer puerto serie del ordenador. Evidentemente, este equipo no está a la última, pero eso es solo para demostrar que Linux es poco exigente con los recursos. La línea 35 configura el `baudrate` a 4800

para asegurar que el dispositivo X10 conectado al puerto serie reciba el mensaje correctamente.

### Potencia de root restringida

*lamp.pl* accede al puerto serie del ordenador y necesita ejecutarse como *root* para hacer esto. La línea 29 comprueba que el *user ID* sea el correcto y detendrá la ejecución si no es 0 (*root*). Como el cliente Jabber se ejecuta con un *user ID* sin privilegios, el listado *lamp.c* define

un envoltorio C, que se puede compilar así de fácil:

```
gcc -o lamp lamp.c
```

Ajustando el bit de *setuid* vía *chmod 4755 lamp* de manera que un usuario 'normal' pueda ejecutar el binario compilado *lamp* y por tanto el guión Perl */usr/bin/lamp.pl* como *root*:

```
$ ls -l /usr/bin/lamp*
-rwsr-xr-x 1 root root 11548
```

```
Oct 2 08:48 lamp
-rwxr-xr-x 1 root root 742
Oct 2 08:45 lamp.pl
```

En esta configuración, solamente *root* puede modificar el guión *lamp.pl*, pero un usuario normal puede ejecutar *lamp.pl* con el *ID* efectivo de *root*.

### Guardian Bot

Y ahora volvamos al *bot*:

Para evitar que cualquier cliente antiguo de Jabber envíe ordenes,

#### Listado 1: agent.pl

```
001 #!/usr/bin/perl
002 #####
003 # agent -- Jabber bot
004 # Operando tras un
    cortafuegos
005 # Mike Schilli, 2004
006 # (m@perlmeister.com)
007 #####
008 use warnings;
009 use strict;
010
011 use Net::Jabber qw(Client);
012 use Log::Log4perl qw(:easy);
013 use LWP::Simple;
014
015 Log::Log4perl->easy_init(
016 {
017   level => $DEBUG,
018   file =>
019     '>>/tmp/agent.log'
020 }
021 );
022
023 my $JABBER_USER =
024   'mikes-agent-receiver';
025 my $JABBER_PASSWD = "*****";
026 my $JABBER_SERVER =
027   "jabber.org";
028 my $JABBER_PORT = 5222;
029
030 our %ROSTER;
031
032 my $c =
033   Net::Jabber::Client->new();
034
035 $c->SetCallbacks(
036 {
037   message => sub {
038     my $msg = $_[1];
039
040     DEBUG "Mensaje '",
041     $msg->GetBody(),
042     "' from ",
043     $msg->GetFrom();
044
045     if ( !exists
046       $ROSTER{ $msg->GetFrom()
047     } ) {
048       INFO "Denegado (no está en "
049         .
050         "la lista)";
051     return;
052   }
053
054     DEBUG "Ejecutando ",
055     $msg->GetBody();
056     my $rep =
057       run_cmd(
058         $msg->GetBody() );
059     chomp $rep;
060     DEBUG "Resultado: ", $rep;
061
062     $c->Send(
063       $msg->Reply(
064         body => $rep
065       )
066     );
067   },
068   onauth => sub {
069     DEBUG "Auth";
070     %ROSTER =
071       $c->RosterGet();
072     $c->PresenceSend();
073   },
074   presence => sub {
075     # Ignorar todas las
076     # peticiones de subscripción
077   },
078   },
079   },
080 );
081
082 DEBUG "Conectando ...";
083
084 $c->Execute(
085   hostname => $JABBER_SERVER,
086   username => $JABBER_USER,
087   password => $JABBER_PASSWD,
088   resource => 'Script',
089 );
090
091 $c->Disconnect();
092
093 #####
094 sub run_cmd {
095   #####
096   my ($cmd) = @_;
097
098   # Encontrar IP Externo
099   if ( $cmd eq "ip" ) {
100     return LWP::Simple::get(
101       "http://perlmeister" .
102       ".com/cgi/whatsmyip"
103     );
104   }
105
106   # Imprimir Carga
107   if ( $cmd eq "load" ) {
108     return `usr/bin/uptime`;
109   }
110
111   # Encender/Apagar luz
112   if ( $cmd =~
113     /^!lamp\s+(on|off)$/ )
114   {
115     my $src =
116       system(
117         "usr/bin/lamp $1");
118     return $src == 0
119       ? "ok"
120       : "not ok ($src)";
121   }
122
123   return "Orden desconocida";
124 }
```

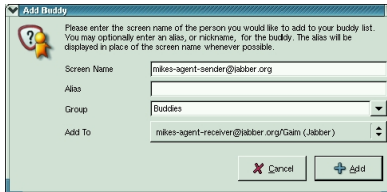


Figura 7: Usando el cliente *gaim* para añadir el agente de ordenes a la lista de emisores aceptados.



Figura 8: Añadiendo el emisor a la lista de conocidos del "bot".

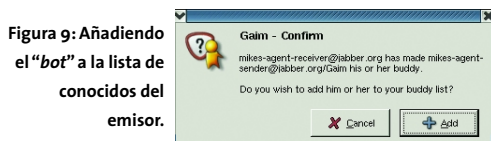


Figura 9: Añadiendo el "bot" a la lista de conocidos del emisor.

*agent.pl* solo acepta mensajes de las personas de su lista de nombres. Cuando llega un mensaje, la línea 45 comprueba que el emisor está en la lista y lo rechazará si no es así.

El manejador de petición de presencia definido en la línea 75 está vacío e ignora cualquier petición desde clientes que quieran añadir al agente en su propia lista de conocidos. *Net::Jabber::Client* trae un manejador predeterminado que

acepta mensajes *presence* desde cualquier cliente de la red. Esto no sería tan malo, pero sin más dificultades, pondrá a estos clientes en su lista de conocidos. Un manejador de *presence* vacío evitará que ocurra esto.

## Instalación

Cuando se instale el *bot*, se debe configurar su lista de conocidos. La mejor manera de hacerlo es usando un cliente *gaim* [4], creando 2 nuevas cuentas en Jabber, *mikes-agent-receiver* y *mikes-agent-sender* y hacer que *mikes-agent-receiver* tenga a *mikes-agent-sender* en su lista de conocidos (Véase la Figura 7).

Si ambas cuentas están en línea, en *mikes-agent-sender* emergerá el cuadro de dialogo mostrado en la Figura 8 y será necesario pulsar en *Authorize* para indicar al servidor que permita que se realice la acción. Entonces se preguntara a *mikes-agent-sender* si quiere poner a *mikes-agent-receiver* en su lista de conocidos (Figura 9); por supuesto que sí, hacer esto tiene sentido para permitir al remitente seleccionar el nombre en la lista de conocidos para enviar una orden al *bot*.

## Listado 2: lamp.pl

```

01 #!/usr/bin/perl
02 #####
03 # lamp -- x10 light switch
04 # Mike Schilli, 2004
05 # (m@perlmeister.com)
06 #####
07
08 use warnings;
09 use strict;
10
11 use Device::SerialPort;
12 use ControlX10::CM11;
13
14 my $UNIT_CODE = "F";
15 my $HOUSE_CODE = "1";
16
17 my %cmds = (
18 "on" => "J",
19 "off" => "K",
20 );
21
22 die "utilización: $0 [on|off]"
23 if @ARGV != 1
24 or $ARGV[0] !~
25 /^(on|off)$/;
26
27 my $onoff = $1;
28
29 die "Debe ser root"
30 if $> != 0;
31
32 my $serial =
33 Device::SerialPort->new(
34 '/dev/ttyS0', undef );
35 $serial->baudrate(4800);
36
37 # Unidad de Dirección
38 ControlX10::CM11::send(
39 $serial,
40 $UNIT_CODE . $HOUSE_CODE );
41
42 # Enviar Instrucción
43 ControlX10::CM11::send(
44 $serial,
45 $UNIT_CODE . $cmds{$onoff}
46 );

```

## Listado 3: lamp.c

```

01 main(int argc, char **argv) {
02   exevc("/usr/bin/lamp.pl",
03   argv);
04 }

```

Después de salir de la sesión, hay que asegurarse de que la cuenta *mikes-agent-receiver* solo se utilice por *agent.pl* y no por otros clientes, para evitar que enreden en la lista de conocidos, que proporciona el mecanismo de la autorización.

Cuando se lance *agent.pl*, *mikes-agent-receiver* debe aparecer en la lista de conocidos de *mikes-agent-sender* (Véase la Figura 2). El archivo de bitácora */tmp/agent.log*, anotará lo sucedido en el caso de que sea necesario depurar la configuración.

Hay que ser cuidadoso cuando se aplican modificaciones, un pequeño error en la implementación puede abrir un agujero en el cortafuegos ¡Hay que vigilarlo!

Desde luego que si se vive en una zona donde la infraestructura eléctrica no da soporte a X10, se necesitará encontrar otra manera de comunicar con el interruptor de la lámpara. Pero, aparte de este proyecto, estas técnicas ayudarán a comenzar con la construcción de un agente de mensajería instantánea. ■

## RECURSOS

- [1] Listados de este artículo: <http://www.linuxmagazine.com.es/Magazine/Downloads/o3>
- [2] "Jabber Developer's Handbook" de Dana Moore y William Wright, Colección Developer's Library, de la editorial Sam's Publishing, 2004.
- [3] Dispositivos X10 de: <http://x10.com>
- [4] Gaim, el cliente de mensajería instantánea universal: <http://gaim.sourceforge.net>
- [5] Dispositivos Domóticos: <http://www.domotica.net/>

## EL AUTOR

Michael Schilli trabaja como desarrollador de software en Yahoo!, Sunnyvale, California. Es el autor de "Perl Power" de la editorial Addison-Wesley y se le puede contactar en [mschilli@perlmeister.com](mailto:mschilli@perlmeister.com). Su página está en <http://perlmeister.com>.