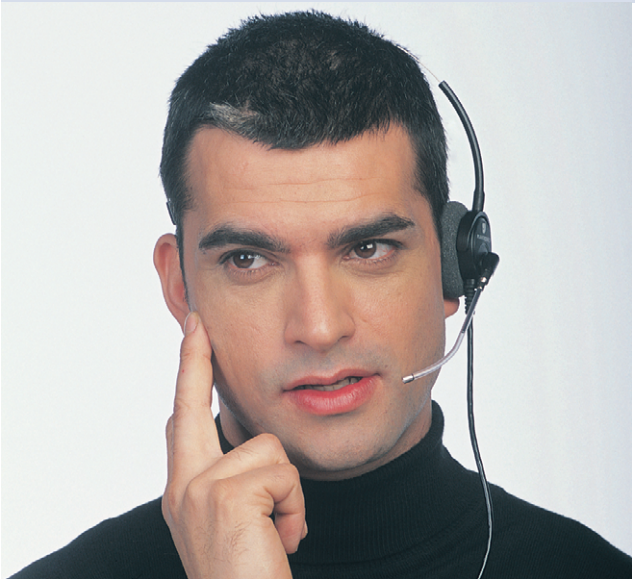
Fernsteuerung via Instant Messenger

Kontaktmann

Wer von außen durch die eigene Firewall ins LAN gelangen will, braucht entweder ein geheimes Loch oder einen kooperativen Agenten auf der Innenseite. Ein Jabber-Client nimmt von innen aktiv Kontakt zum öffentlichen Jabber-Server auf und wartet auf Instruktionen seiner Internet-Buddies in Form von Instant Messages. Michael Schilli



Um vom Internet aus Aktionen im lokalen Netzwerk auszulösen, könnte man ein Loch in die hoffentlich vorhandene Firewall bohren und einen lokalen Webserver ins Internet stellen. Dynamisch vergebene IP-Adressen des Internetproviders verfolgen Services wie Dyndns .org und erlauben einen beinahe statischen Zugriff. Einfacher geht's heute allerdings mit einem Agenten oder Bot (wohl von Robot): Ein auf der Innenseite der Firewall laufender Client hängt im öffentlichen Instant-Messaging-Netzwerk Jabber und versteht Kommandos, die ihm in Form von Textnachrichten übermittelt werden.

Der Bot akzeptiert nur Befehle von Clients aus seiner Buddy-Liste und erlaubt lediglich vier Aktionen: die Last auf dem Agenten-Rechner bestimmen, die öffentliche Adresse des Routers abfragen (Kommando: »ip«) und das Licht im Schlafzimmer meiner Wohnung in San Francisco ein- beziehungsweise ausschalten (»lamp on|off«).

Das Skript »agent.pl« nutzt Log::Log4perl und führt in der Datei »/tmp/agent.log« Buch über die ausgeführten Transaktionen. Die Zeile 24 in Listing 1 erzeugt ein neues »Net::Jabber::Client«-Objekt, das einen Instant-Message-Client implementiert.

Rückruf

Bevor aber »agent.pl« in die Haupt-Eventschleife in Zeile 61 eintritt, sind ab Zeile 26 noch einige Callbacks für eintretende Ereignisse zu definieren. Der »onauth«-Handler ab Zeile 48 kommt an die Reihe, wenn der Client sich erfolgreich mit dem in Zeile 17 angegebenen Nutzernamen beim Server angemeldet hat. Der Handler fragt dann mittels »RosterGet()« die Buddy-List ab und speichert sie im globalen Hash »%ROSTER«.

Die anschließend abgesetzte Methode »Presence()« schickt allen Clients im Roster eine »presence«-Nachricht, die ihnen anzeigt, dass »agent.pl« online ist. Ab diesem Zeitpunkt zeigt ein »gaim«-Client mit »mikes-agent-sender« als eingeloggtem Benutzer »mikes-agent-receiver« als aktiven Client in seiner Buddy-List an (Abbildung 3).

Der »message«-Callback ab Zeile 28 kommt zum Zuge, wenn der Kommandeur eine Nachricht an »agent.pl« schickt. Jeder beliebige Client im Jabber-Netzwerk könnte dies tun. Deshalb prüft Zeile 34, ob es sich beim Gesprächspartner auch um einen bekannten und berechtigten handelt. Im vorliegenden Fall darf es nur »mikes-agent-sender« sein, da die Buddy-List des Clients nur ihn enthält (siehe Abschnitt „Installation"). Andere Anfrager lässt die Funktion abblitzen und kehrt in Zeile 37 zur Hauptschleife zurück.

Endlos

Das im Message-Text geschickte Steuerungskommando kramt in Zeile 41 die »GetBody()«-Methode hervor und gibt sie an die ab Zeile 71 definierte »run_cmd«-Funktion weiter. »Execute()« ab Zeile 61 nimmt Verbindung mit dem Jabber-Server auf »Jabber.org« auf und meldet »mikes-agent-receiver« dort an. Die Haupt-Eventschleife erholt sich auch bei temporär abreißender Verbindung und sollte niemals enden. Falls sie jedoch nach zu vielen Fehlern dennoch abbricht, räumt Zeile 68 auf und das Programm endet.

Von hinten durch die Brust ins Auge

Die IP-Adresse meines Routers findet der Agent heraus, indem er einen Web-Request auf eine öffentlich zugängliche URL abfeuert: [http://perlmeister.com/cgi/whatsmyip]. Dort hängt nur ein ein-

Der Autor

Michael Schilli arbeitet als Software-Engineer bei Yahoo! in Sunnyvale, Kalifornien. Er hat „Goto Perl 5" (deutsch) und „Perl Power" (englisch) für Addison-Wesley geschrieben und ist unter [mschilli@perlmeister.com] zu erreichen. Seine Homepage ist [http://perlmeister.com].

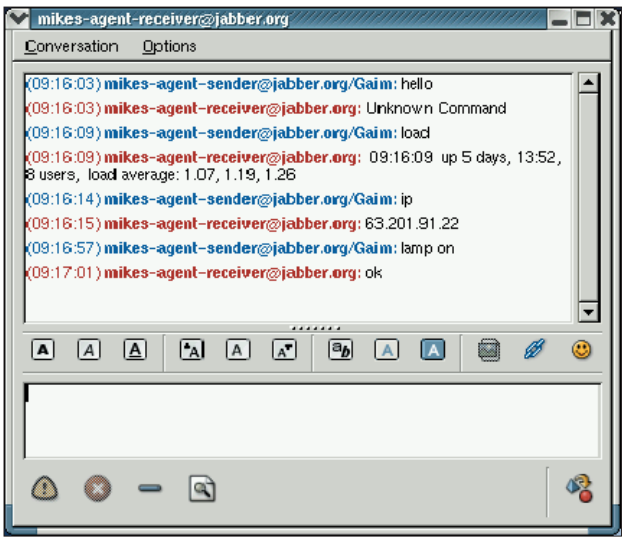


Abbildung 1: Der Bot hinter der Firewall führt Befehle aus, die ein Jabber-Client vom Internet sendet.

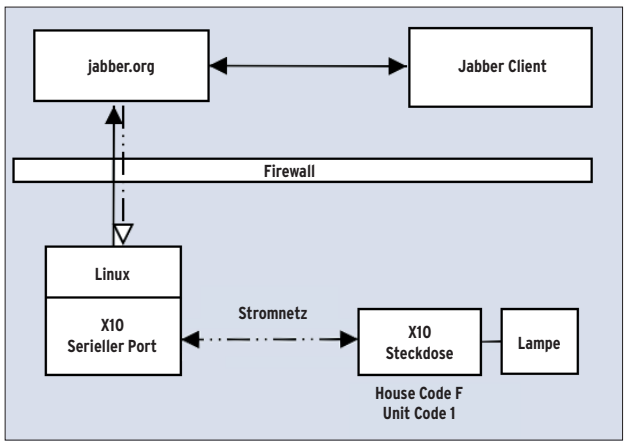


Abbildung 2: Die Internet-gestützte Lampensteuerung in der Übersicht.

faches Skript, das lediglich die Client-Adresse des Aufrufers zurückgibt:

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

»agent.pl« zieht hierzu »LWP::Simple« heran. Dessen »get«-Funktion holt in Zeile 77 den Inhalt der Webseite, falls die empfangene Textnachricht »ip« war. Ähnliches gilt für die aktuelle Rechnerlast: Zeile 84 ruft das Linux-Kommando »uptime« auf und gibt das Ergebnis an Zeile 41 zurück, wo es mit »chomp« zurechtgestutzt, geloggt und schließlich in Zeile 45 in einen Message-Body verpackt und per »Send« an den Chat-Partner zurückgeschickt wird.

Doch wie schaltet der auf einem Linux-Rechner laufende Agent das Schlafzimmerlicht an? In den USA gibt es (natürlich nur für das dortige elektrische System) die X10-Technologie, die Signale über die elektrischen Leitungen im Haus verschickt und auch über eine serielle oder USB-Schnittstelle mit dem Rechner kommunizieren kann.

Jede Kontrolleinheit (Abbildung 4) verfügt über einen House-Code (A bis K) und einen Unit-Code (1 bis 9), den die Steuereinheit selektieren muss, damit auch das richtige Licht (und nicht das des Nachbarn) angeht. Das Ganze ist nicht teuer: Ein vielteiliges Einsteigerkit mit allerhand Schnickschnack und Fernbedienung kostet auf [3] zwischen 50 und 100 Dollar.

Europatauglich

Für europäische Verhältnisse taugt das System nicht, doch es gibt Alternativen. Eine ähnliche Funktion bietet beispielsweise der seriell steuerbare Schalter SI-230, den es bei ELV [5] zu kaufen gibt. Er braucht allerdings eine direkte Kabelverbindung zum Rechner. Wie ein solches Kabel schnell selber herzustellen ist, erklärt [6]. Ein Beispielprogramm für die Steuerung, die sich auch in Perl schnell programmieren lässt, ist unter [7] zu finden.

Listing 2 »lamp.pl« zeigt das kurze Skript, das die Codes über den seriellen Port aussendet und damit die Lampe steuert. Es nutzt lediglich »Device::SerialPort« und »ControlX10::CM11« vom CPAN, um zunächst die richtige Einheit mit House/Unit-Code zu adressieren und dann nochmals den House-Code gefolgt von J (einschalten) oder K (ausschalten) zu schicken. Als serieller Port wählt Zeile 28 »/dev/ttyS0«, da die Steuereinheit im ersten seriellen Port des Rechners steckt. Ich gebe zu, mein Rechner ist nicht der neueste, aber Linux ist ja genügsam ▶.

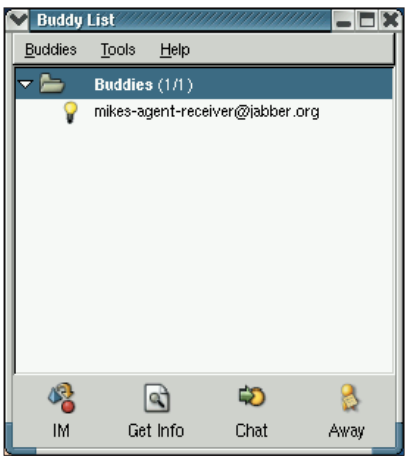


Abbildung 3: Der Bot erscheint in der Buddy-List des Senders.

Abbildung 4: Die X10-Kontrolleinheit wartet auf Signale und schaltet den Strom an oder ab.

Abbildung 5: Die Schlafzimmerlampe wurde übers Internet angeschaltet.

»lamp.pl« greift auf den seriellen Port des Computers zu und muss daher als Root laufen. Zeile 25 prüft die effektive User-ID und bricht das Programm ab, falls sie nicht der Root gehörenden Nummer »0« entspricht. Da der Jabber-Client unter einem minder privilegierten Nutzer läuft, definiert der Code

```
main(int argc, char **argv) {
    execv("/usr/bin/lamp.pl", argv);
}
```

einen C-Wrapper, der einfach folgendermaßen übersetzt wird:

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

Root-Gefängnis

Mit gesetztem Setuid-Bit kann ein normaler Nutzer dann das Perl-Skript »/usr/bin/lamp.pl« unter der effektiven ID »root« ablaufen lassen:

```
$ 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
```

In dieser Konfiguration ist es nur »root« möglich, das Skript »lamp.pl« zu verändern.

Agenten-Sicherheit

Zurück zum Agenten: Damit nicht jeder dahergelaufene Jabber-Client Befehle senden kann, akzeptiert »agent.pl« nur Nachrichten von Leuten auf seinem Jabber-Roster. Bei eingehenden Nachrichten prüft Zeile 34 (Listing 1), ob der Sender auf der Liste steht, und verweigert sonst den Zugriff.

Der ab Zeile 54 definierte Handler für »Presence«-Requests ist leer und ignoriert alle Anfragen von Clients, die den Agenten auf ihre Buddy-Liste setzen wollen. Das wäre zwar noch nicht so schlimm, doch der von »Net::Jabber« bereitgestellte Default-Handler akzeptiert solche Anfragen nicht nur, sondern setzt anfragende Clients sofort auf seine eigene Buddy-Liste. Ein leerer »presence«-Handler verhindert dies.

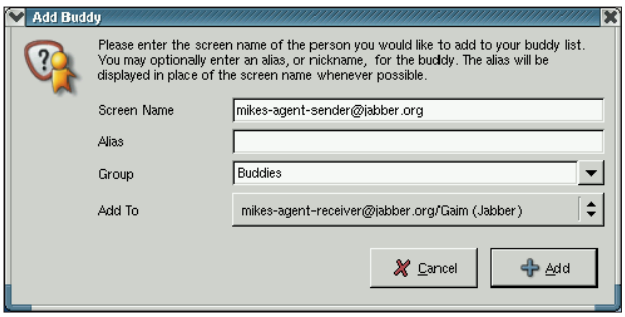


Abbildung 6: Ein Gaim-Client nimmt den befehlenden Agenten in die Liste der erlaubten Sender auf.

Installation

Bei der Installation des Agenten muss dessen Buddy-Liste gesetzt werden, am besten mit einem »gaim«-Client [4], der unter Angabe des Passworts einen neuen Account »mikes-agent-receiver« anlegt, sich dort einloggt und den ebenfalls frisch angelegten »mikes-agent-sender« auf die Liste setzt (Abbildung 6).

Listing 1: »agent.pl«

```
01 #!/usr/bin/perl
02 ##########################################
03 # agent - Jabber agent behind firewall
04 # Mike Schilli, 2004 (m@perlmeister.com)
05 ##########################################
06 use warnings;
07 use strict;
08
09 use Net::Jabber qw(Client);
10 use Log::Log4perl qw(:easy);
11 use LWP::Simple;
12
13 Log::Log4perl->easy_init(
14     { level => $DEBUG,
15       file => '>>/tmp/agent.log' });
16
17 my $JABBER_USER   = 'mikes-agent-receiver';
18 my $JABBER_PASSWD = "******";
19 my $JABBER_SERVER = "jabber.org";
20 my $JABBER_PORT   = 5222;
21
22 our %ROSTER;
23
24 my $c = Net::Jabber::Client->new();
25
26 $c->SetCallBacks(
27
28   message => sub {
29     my $msg = $_[1];
30
31     DEBUG "Message '", $msg->GetBody(),
32           "' from ", $msg->GetFrom();
33
34     if(! exists
35        $ROSTER{$msg->GetFrom()}) {
36         INFO "Denied (not in roster)";
37         return;
38     }
39
40     DEBUG "Running ", $msg->GetBody();
41     my $rep = run_cmd($msg->GetBody());
42     chomp $rep;
43     DEBUG "Result: ", $rep;
44
45     $c->Send($msg->Reply(body => $rep));
46   },
47
48   onauth => sub {
49     DEBUG "Auth";
50     %ROSTER = $c->RosterGet();
51     $c->PresenceSend();
52   },
53
54   presence => sub {
55     # Ignore all subscription requests
56   },
57 );
58
59 DEBUG "Connecting ...";
60
61 $c->Execute(
62   hostname => $JABBER_SERVER,
63   username => $JABBER_USER,
64   password => $JABBER_PASSWD,
65   resource => 'Script',
66 );
67
68 $c->Disconnect();
69
70 ##########################################
71 sub run_cmd {
72 ##########################################
73   my($cmd) = @_;
74
75     # Find out external IP
76   if($cmd eq "ip") {
77     return LWP::Simple::get(
78     "http://perlmeister.com/cgi/whatsmyip"
79     );
80   }
81
82     # Print Load
83   if($cmd eq "load") {
84     return `/usr/bin/uptime`;
85   }
86
87     # Switch bedroom light on/off
88   if($cmd =~ /^lamp\s+(on|off)$/) {
89     my $rc = system("/usr/bin/lamp $1");
90     return $rc == 0 ? "ok" :
91                       "not ok ($rc)";
92   }
93
94   return "Unknown Command";
95 }
```

Sind beide Accounts online, erscheint für »mikes-agent-sender« ein Dialog, in dem »Authorize« zu klicken ist, damit der Server den Vorgang erlaubt. »mikes-agent-sender« bekommt anschließend noch das Angebot, »mikes-agent-receiver« in seine Buddy-List aufzunehmen, was dieser praktischerweise tut.

Nach dem anschließenden Logout sollte der Account »mikes-agent-receiver« nur noch vom Skript »agent.pl« und nicht mehr von anderen Clients benutzt werden, denn die könnten versehentlich seine Buddy-Liste umkrempeln, die hier zugleich als Autorisierungs-Mechanismus dient. Nach dem Start von »agent.pl« erscheint »mikes-agent-receiver« in der Buddy-Liste von »mikes-agent-sender« (Abbildung 3). Die Logdatei »/tmp/agent.log« protokolliert die einzelnen Schritte.

Zum Schluss ein Appell: Beim Probieren bitte sehr behutsam vorgehen, denn jeder Implementierungsfehler könnte schnell ein gefährliches Loch in die Firewall reißen. *(jcb)* ■

Infos

[1] Listings zum Artikel: [ftp://www.linux-magazin.de/pub/listings/magazin/2004/12/Perl]
[2] Dana Moore and William Wright, „Jabber Developer's Handbook“: Developer's Library, Sam's Publishing, 2004
[3] Vertrieb von X10-Geräten: [http://x10.com]
[4] Gaim, der universelle Instant-Message-Client: [http://gaim.sourceforge.net]
[5] ELV Elektronik: [http://www.elv.de]
[6] Mirko Dölle, „Schaltstelle“: Easy Linux 5/04
[7] Mirko Dölle, „Schnell geschaltet“: Linux-Magazin 9/00

Listing 2: »lamp.pl«

```
01 #!/usr/bin/perl
02 ##########################################
03 # lamp -- Switch lamp on and off via x10
04 # Mike Schilli, 2004 (m@perlmeister.com)
05 ##########################################
06 use warnings;
07 use strict;
08
09 use Device::SerialPort;
10 use ControlX10::CM11;
11
12 my $UNIT_CODE  = "F";
13 my $HOUSE_CODE = "1";
14
15 my %cmds = (
16     "on"  => "J",
17     "off" => "K",
18 );
19
20 die "usage: $0 [on|off]" if @ARGV != 1
21     or $ARGV[0] !~ /^(on|off)$/;
22
23 my $onoff = $1;
24
25 die "You must be root" if $> != 0;
26
27 my $serial = Device::SerialPort->new(
28     '/dev/ttyS0', undef);
29 $serial->baudrate(4800);
30
31     # Adress unit
32 ControlX10::CM11::send($serial,
33     $UNIT_CODE . $HOUSE_CODE);
34
35     # Send command
36 ControlX10::CM11::send($serial,
37     $UNIT_CODE . $cmds{$onoff});
```

User Friendly – der monatliche Comic im Linux-Magazin