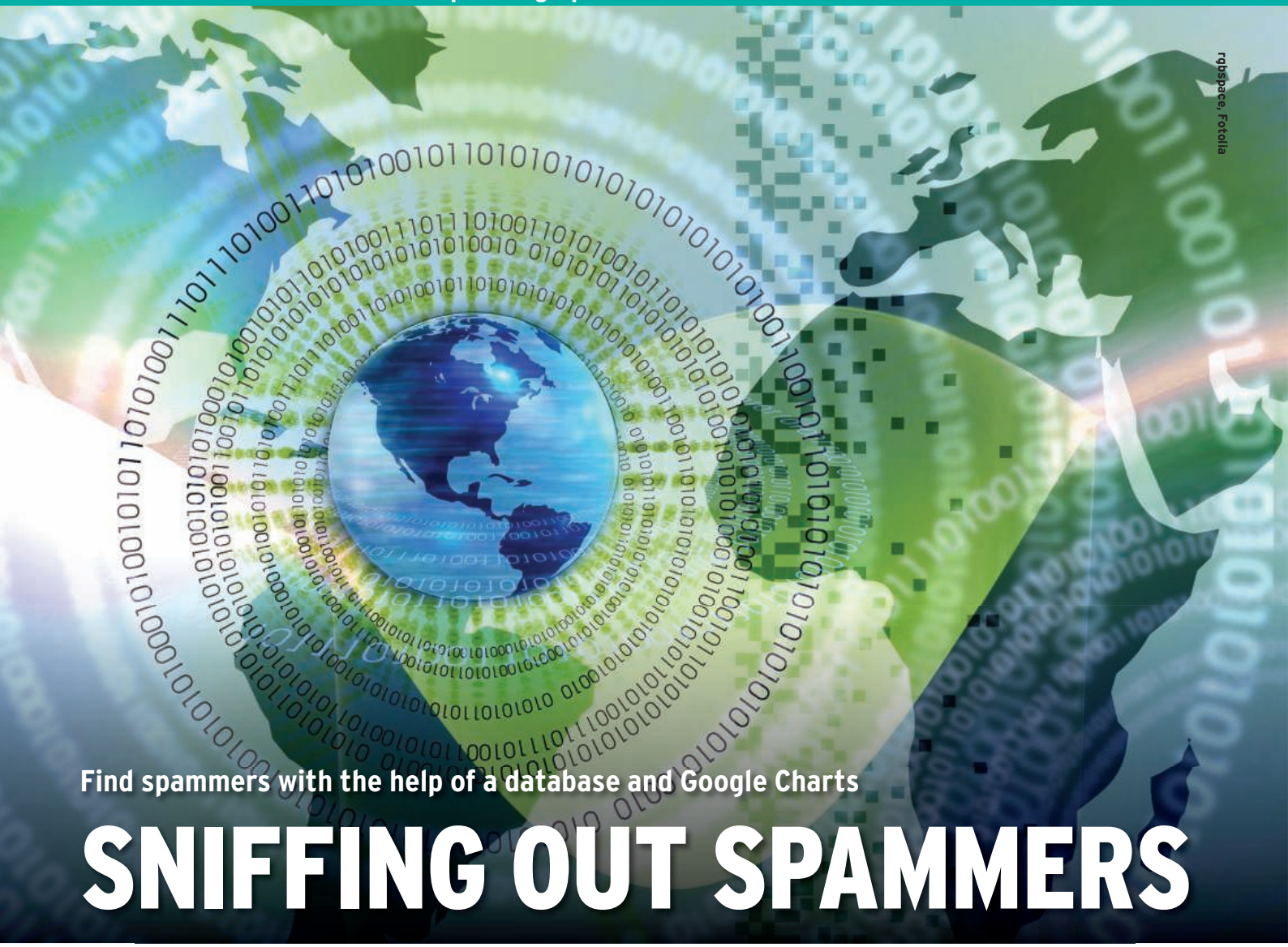

Find spammers with the help of a database and Google Charts

SNIFFING OUT SPAMMERS

To identify the geographic regions from which link spam originated, a database locates IP addresses and the Google Charts service puts them onto a world map. **BY MICHAEL SCHILLI**

Sometimes I imagine how satisfying it would be to track down a spammer or telemarketer's office, like the man in the Snickers commercial [1] who arrives at the office and gets his revenge. Unfortunately, legal and logistical reasons often prevent this. Additionally, it is often the case that the perpetrators are botnets rather than the spammers themselves. Still, it would be interesting to create a graph that pinpoints the geographic regions in which most spam activities originate.

The Internet is the ideal platform for anonymous trickery, but the perpetrator's deeds actually leave a trail – each incoming request on a website includes the sender's IP address (see Figure 1).

Of course, the address could be spoofed, but this is not so simple and just too much trouble for most link spammers.

The DNS system, which resolves hostnames into IP addresses, can often do the same thing in reverse gear. A DNS reverse lookup expects an IP address, and if the spammer's service provider has set up everything as it should be, the script in Listing 1, *revlookup*, will return a hostname from which you usually can identify the provider. Figure 2 shows that the address I caught spamming, IP 69.162.110.146, belongs to an ISP called *lstn.net*; a friendly email to the ISP's webmaster, stating the IP and the time (which is important because these IPs are often assigned dynamically) might just be the ticket to stop the spammer's illegal activity for good.

The *inet_aton()* function in Perl's *Socket* module accepts an IP address in string notation ("*x.x.x.x*") and returns a data structure for a subsequent call to Perl's *gethostbyaddr()* function. When called with the *AF_INET* parameter, as shown in line 17 in *revlookup*, the func-

```
access.log + (~/DEV/articles/charts/eg) - VIM
93.135.197.130 - - [26/Oct/2008:12:29:53 -0700] "GET /76/images/chilkoot.jpg HTT
P/1.1" 200 107901 "Mozilla/5.0 (X11; U; Linux x86_64; en-US; rv:1.9.0.3) Gecko/2
008092510 Ubuntu/8.04 (hardy) Firefox/3.0.3"
69.162.110.146 - - [26/Oct/2008:12:30:03 -0700] "GET /forum/posting.php?mode=new
topic&f=1 HTTP/1.0" 200 50299 "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; r
v:1.7) Gecko/20041122 Firefox/0.5.6+"
69.162.110.146 - - [26/Oct/2008:12:30:10 -0700] "POST /forum/posting.php?sid=77f
80b8c8996b56f3dd43314eef8f422 HTTP/1.0" 200 29999 "Mozilla/5.0 (Windows; U; Wind
ows NT 5.1; en-US; rv:1.7) Gecko/20041122 Firefox/0.5.6+"
"access.log" [Modified] 3 lines --33%--                    2,1           All
```

Figure 1: Link spammers leave their IP addresses in the web server's access log.

Listing 1: revlookup

```
01 #!/usr/local/bin/perl -w
02 use strict;
03 use Socket;
04
05 my $host = $ARGV[0]
06   or die "usage: $0 ipaddr";
07
08 print reverse_lookup($host)
09   || "unknown", "\n";
10
11 ############################
12 sub reverse_lookup {
13 ############################
14  my ($ip) = inet_aton $_[0];
15
16  return (
17   gethostbyaddr(
18    $ip, AF_INET
19   )
20  )[0];
21 }
```

tion performs the DNS reverse lookup in the IPv4 address space and, if successful, returns a string with the hostname or returns *undef* if an error occurs. Depending on how busy the DNS server you call on is at the time and how many of its peers it needs to consult to answer your request, this process can take a couple of seconds.

As another option, the *whois* command-line utility doesn't just work with domains, it also accepts IP addresses as arguments. Figure 3 shows that the provider, Limestone Networks, has registered everything correctly and even provides an email address that spammed webmasters can contact with their complaints. The lookup can be automated in Perl with the CPAN *Net::Whois::Raw* module, for example; however, it puts significant load on the servers hosted by Network Solutions, who will block access if you perform 100 lookups in quick succession. In other words, searching a complete access log with this module is impossible, even if you cache queries you have already made.

Many spammers use IP addresses without a reverse lookup entry on the DNS system. But even then you can still locate the culprit; IP addresses are assigned to service providers in blocks, and you can download databases with the information necessary to discover the approximate geographic position of any given IP address. MaxMind offers a database file [2] that is free for non-commercial use. The licensing conditions are available in the same directory as the database itself. The CPAN *IP::Country::MaxMind* module provides an API to match, thus avoiding the need to mess around with data blobs. The IP mappings stored in the database change very slowly; updating once every couple of months should be fine.

After installing the module, you will need another CPAN module, *Geo::IP::PurePerl*. The MaxMind module's *open()* constructor loads the local database that you specify, and the *inet_atocc()* function returns a country code for any IP address (for example, DE for Germany).

The Google Charts API [3] gives you a useful option for plotting these codes on a world map. If you pass in pairs of values to the Google server, it will respond with a PNG-formatted image file. The data format for the pairs of values is slightly unusual in that you need to squash largish volumes of data into the very restricted space offered by a URL and its query parameters.

Simplicity Itself

The API's aptly named Simple Encoding data format will only allow values between 0 and 61, encoded as A-Z (0-25), a-z (26-51), and 0-9 (52-61).

If you assign a value of 23 to Germany, 3 to the USA, and 60 to Japan, you can encode the country codes in the *chld* URL parameter as "*DEUSJP*" (DE, US, and JP, concatenated without blanks), and the values as "*s:XD8*" (s = simple encoding, X = 23, D = 3, and 8 = 60) in *chd*.

Figure 2: A reverse DNS lookup often reveals the domain associated with the IP address.

The script in Listing 2, *spam2geo*, implements the steps I have identified thus far; it analyzes the *access.log* file from an Apache server under heavy fire from link spammers. The CPAN *ApacheLog::Parser* module provides a *parse_line_to_hash* function, which understands the *access.log* format and returns the individual fields of each log entry as a hash. The *client* entry includes the spammer's IP address in each case, and a call to the *inet_atocc* method in line 32 returns the two-letter country code, assuming the database knows it.

If successful, line 36 increments the hash entry for the country, and the program moves on to the next line in the logfile. Because you are not interested in all the URLs – just the ones generated by spammers – line 28 filters out all entries whose path (*file* hash key) does not match the regular expression *posting*. The regex should only match URLs used by spammers to post on the forums you are monitoring, so you must modify it to match your local conditions.

Listing 2: spam2geo

```
001 #!/usr/local/bin/perl -w
002 use strict;
003 use LWP::UserAgent;
004 use URI::URL;
005 use List::Util qw(max min);
006
007 use IP::Country::MaxMind;
008 use ApacheLog::Parser
009   qw(parse_line_to_hash);
010
011 my $gi =
012   IP::Country::MaxMind->open(
013  "GeoIP.dat");
014
015 my %by_country;
016
017 open LOG, "<access.log"
018   or die
019 "Can't open access.log ($!)";
020
021 while (<LOG>) {
022  chomp;
023  my %fields =
024    parse_line_to_hash $_;
025
026  # only proceed if forum post
027  next
028    if $fields{file} !~
029    /posting/;
030
031  my $country =
032    $gi->inet_atocc(
033   $fields{client});
034
035  if (defined $country) {
036   $by_country{$country}++;
037  }
038 }
039
040 close LOG;
041
042 # Convert values to Google format
043 my @SYMBOLS = (
044  "A" .. "Z",
045  "a" .. "z",
046  0 .. 9
047 );
048
049 my $max =
050   max values %by_country;
051 my $min =
052   min values %by_country;
053
054 for my $country (
055  keys %by_country)
056 {
057
058  my $val =
059    $by_country{$country};
060  my $norm =
061    ($val - $min) / $max *
062    $#SYMBOLS;
063
064  $by_country{$country} =
065    $norm;
066 }
067
068 my $chld = join "",
069   keys %by_country;
070 my $data = join "",
071   values %by_country;
072
073 # Fetch chart
074 my $ua =
075   LWP::UserAgent->new();
076
077 my $uri =
078   URI::URL->new(
079 "http://chart.apis.google.com/chart"
080   );
081
082 $uri->query_form(
083  cht  => "t",
084  chs  => "440x220",
085  chtm => "world",
086  chd  => "s:$data",
087
088  # white, yellow, red
089  chco =>
090    "ffffff,f4ed28,f11414",
091  chld => $chld,
092
093  # light blue
094  chf => "bg,s,EAF7FE"
095 );
096
097 my $resp = $ua->get($uri);
098
099 # Print image on success
100 if ($resp->is_success()) {
101
102  open FILE, ">file.png"
103    or die;
104  print FILE $resp->content();
105  close FILE;
106  system("eog file.png");
107 } else {
108
109  die $resp->request->url()
110    . " failed\n";
111 }
```

Anzeige
wird
separat
angeliefert

Normalization and conversion of the data to the Google format starts in line 42. Because the numeric values for each country in the *%by_country* hash are not necessarily in the range 0–61 but can assume arbitrary values, *spam2geo* must determine the limits of the range by use of *min* and *max* from *List::Util*. After doing so, it subtracts *$min* and divides by *$max* to squash the numeric values to be represented into the range between 0 and 1 and multiplies the latter value by the number of encoding characters minus 1. Thus, *$norm* contains a floating point number, which can be converted to an integer and used as an index in the *@SYMBOLS* array, thus mapping the whole range of values to an element in the array.

```
mschilli@mybox:/mnt/data/big1/mschilli.do.not.delete/DEV/articles/ch
$ whois 69.162.110.146

OrgName:    Limestone Networks, Inc.
OrgID:      LIMES-2
Address:    400 N. St. Paul
City:       Dallas
StateProv:  TX
PostalCode: 75201
Country:    US

ReferralServer: rwhois://rwhois.limestonenetworks.com:4321

NetRange:   69.162.64.0 - 69.162.127.255
CIDR:       69.162.64.0/18
OriginAS:   AS30008
NetName:    LSN-DLLSTX-2
NetHandle:  NET-69-162-64-0-1
Parent:     NET-69-0-0-0-0
NetType:    Direct Allocation
NameServer: NS1.LIMESTONENETWORKS.COM
NameServer: NS2.LIMESTONENETWORKS.COM
NameServer: NS3.LIMESTONENETWORKS.COM
Comment:    http://www.limestonenetworks.com
RegDate:    2008-06-27
Updated:    2008-06-27

RAbuseHandle: ABUSE1804-ARIN
RAbuseName:   Abuse
RAbusePhone:  +1-214-586-0555
RAbuseEmail:  abuse@limestonenetworks.com
$
```

Figure 3: The Whois entry for the IP address I caught spamming shows the spammer's Internet provider details.

Lines 68 and 70 then concatenate the calculated symbols to give strings without separating blanks, for passing in with the *chld* (country codes) and *chd* (values) URL parameters. From the programmer's point of view, the order in which the *keys* and *values* functions return results is arbitrary, but consistent within the Perl script, and irrelevant to the Google service.

Communications with the Google server are handled by *LWP::UserAgent* via the http protocol. The URL parameters are set by the *query_form()* method, which also performs any URL encoding required. The *cht* parameter specifies the charts type used by the Google Charts service and is set to "*t*" (topological) for a world map. You can optionally restrict the view to individual continents; however, you need to set the *chtm* parameter to "*world*" for a world map.

The *chs* parameter sets the dimensions of the resulting image to 440x220 pixels. Google Charts uses the colors white, yellow, and red specified as hex RGB values in *chco* to shade the countries, thus reflecting minimum, medium, and maximum values. So, the settings in Listing 2 leave countries with normalized spam counts around 0 white, values of around 20 yellow, and values of 60 or more red. The "*bg,s,EAF7FE*" string for the *chf* parameter stands for background, solid, and the hex value for light blue to color the world's oceans.

All told, the URL will look something like this: *http://chart.apis.google.com/chart?cht=t&chs=440x220&chtm=world&chd=s%3ABFAABAHGQAAA8BAAAAAAaBAA&chco=ffffff%2Cf4ed28%2Cf11414&chld=GBNLHKEELVKRRUSAPAMDCASECNDEPKITPLINMEBRCZUSUAESFR&chf=bg%2Cs%2CEAF7FE*.

Google takes just a couple of seconds to render and deliver this as the graph shown in Figure 4. If you comment out lines 27-29 in *spam2geo*, the graph will give you a geographic distribution of all incoming URLs instead (Figure 5).

Although most spam requests originate in China and the US, most of the website's bona fide customers come from Germany. The *eog file.png* command displays the file produced by Google and retrieved via a web request in the Eye of Gnome utility.

Installation

After downloading the MaxMind *GeoIP.dat.gz* database [2], unpack the *GeoIP.dat* file and place the *spam2geo* script into your current working directory. The CPAN *IP::Country::MaxMind*, *Geo::IP::PurePerl*, *List::Util*, and *ApacheLog::Parser* modules and all their dependencies are best installed from a CPAN shell. To use the Google API, you do not need to register. You just need to modify line 28 in *spam2geo* to match your local conditions by changing the */posting/* pattern to match URLs used only by spammers to clutter your discussion groups with parasitic entries.

For more detailed analysis including, for example, the number of forum requests compared with other activities or the preferred browser type used by the spammers (at least what they say they're using), check out the enormous choice provided by the Google Charts API [3], which gives you an easy approach to render any statistical information elegantly in polished chart form. ■

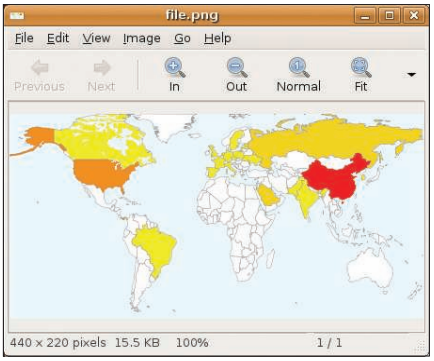

Figure 4: Spammers mainly come from China and North America.

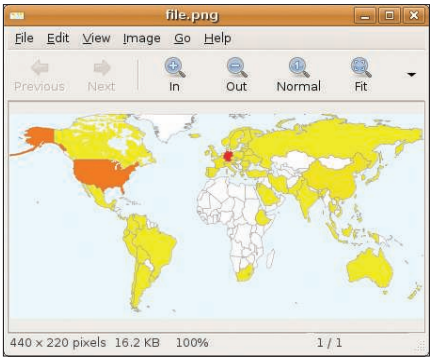

Figure 5: Users of the website mainly come from Germany.

INFO

[1] Snickers Cruncher – Telemarketer: *http://www.youtube.com/watch?v=R6QATC2C0h8*

[2] Free MaxMind GeoIP database download: *http://www.maxmind.com/download/geoip/database/*

[3] "Maps" charts by the Google Charts web service: *http://code.google.com/apis/chart/types.html#maps*

[4] Listings for this article: *http://www.linux-magazine.com/resources/article_code*