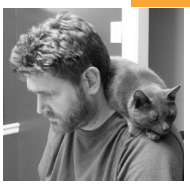
DAVE JOSEPHSEN

iVoyeur: pockets-o-packets, part 3



Dave Josephsen is the author of *Building a Monitoring Infrastructure with Nagios* (Prentice Hall PTR, 2007) and is senior systems engineer at DBG, Inc., where he maintains a gaggle of geographically dispersed server farms. He won LISA '04's Best Paper award for his co-authored work on spam mitigation, and he donates his spare time to the SourceMage GNU Linux Project.

dave-usenix@skeptech.org

THIS MONTH BRINGS YOU THE THIRD and final installment in my series on Argus, the network flow monitoring and reporting framework. If you didn't catch my first two articles (see June and August issues of *;login:*), you should pull them out of your hamster cage and read what's left of them. I covered PCAP hardware, infrastructure, and I also flamed up on the entire database administration profession in general. In this final article on Argus, I'm going to cover the intricacies of the Argus client utilities, and I'll also probably flame up on the DBAs a little more.

So let me give you a feel for how I use Argus on a day-to-day basis. There are myriad reasons I find myself turning to Argus daily, but two spring to mind as especially common: complaints of network slowness, and daily security reports.

"Network slowness" is, as we both know, usually a PEBCAK [1] problem, but when I do want to get a quick feel for what "the network" is doing, I turn to ratop [2]. ratop is a great little utility that implements a network "top" command using Argus data-flows for input. You can point it at files with -r or -R, but I'll usually just go ahead and point it straight at one of our radium streams with -S. This way, I can log straight into the router and fire it up with:

```
ratop -S localhost:4300
```

and get live utilization info similar to this:

```
ratop -S 10.20.0.2:4300200/07/26.20:49:56 CDT
Rank  StartTime        Flgs  Proto  SrcAddr  Sport      Dir  DstAddr  Dport      TotPkts  TotBytes  State
   1  20:49:37.426974  e     tcp    10.20.34.1.40405    ->   10.20.5.1.ssh      27688    3045752   RST
   2  20:49:36.454273  e     tcp    10.20.1.5.57465     ->   10.20.0.2.4300     80       10832     CON
   3  20:49:39.094155  e     vrrp   10.100.1.20         ->   224.0.0.18         12       840       INT
   5  20:49:35.698224  *     llc    0:a:f4:20:60:85.stp ->   1:80:c2:0:0:0.stp  9        540       INT
   6  20:49:37.779319  e     icmp   10.20.68.2          ->   10.100.1.10       3        342       URP
   7  20:49:42.364118  e     tcp    10.25.20.6.55616    <?>  10.20.5.1.ssh      3        258       CON
   8  20:49:43.094867  e     udp    10.100.1.20.42033   <->  10.100.1.10.domain 2        180       CON
   9  20:49:43.095310  e     udp    10.20.4.1.1679      <->  8.15.14.7.domain   2        180       CON
```

OK, this is pretty obvious stuff, but Argus gets even more impressive when you start getting into the data-mining clients, and among these, racluster [3] is king. But rather than bore you (more than I already have) with lists of options, let me tell you a story about our daily reports. We have several cron

jobs that gather security-relevant information from various systems detailing things like people logging into things, files changing, and IDSes being paranoid, so these can be compiled and emailed out in a daily report format. Among these are a few racluster queries that I find useful. These are things like "chatty kathys" (the top ten bandwidth users):

```
racluster -r <yesterdays_file> -w - -m saddr - 'src net 10.201.16.0/21' \
| rasort -m bytes load - -s saddr bytes load  | head -n10
```

"town criers" (top ten broadcasters):

```
racluster -r <yesterdays_file> -w - -m saddr - 'src net <heathen_subnet> \
and dst host <heathen_subnet_broadcast_addr>'  | rasort -m bytes load - \
-s saddr dport bytes load  | head -n10
```

and "the red light district" (top 10 destination ports by total bytes transferred):

```
racluster -r argus-2010-05-20.log -w - -m proto dport \
| rasort -m bytes load - -s dport bytes load  | head -n10
```

If you managed to stay awake through my last article, then some of these options should already be familiar to you, so I'll go ahead and summarize them again to make absolutely sure you won't stay conscious this time.

-r tells racluster the name of an input file. Popular alternatives are to recursively read a directory full of files with -R, or to read directly from an argus daemon or radium process with -S.

-w tells racluster to write out binary data format to STDOUT. If I had specified -w foo, it would have written to a file called foo instead. If no -w is provided, racluster will write human-readable text output. Some Argus tools like rastream have "special" -w options that can do variable expansion for things like writing to date-stamped files.

The story I want to tell you is about the last report I mentioned above (the red-light district). Usually this report looks something like this:

```
80      780506507   730813
22      684953405   675527
443     619322910   423802
8080    536904140   491149
...
```

This is a list of popular destination ports as measured by total byte count and, secondarily, by "load" or bits per second. We use racluster to get a list of all the ports and all the aggregated byte-counts and packet rates for those ports, and then we use rasort to sort the list and give us formatted output.

The -m switch is very important and a little confusing, because it performs related but very different functions in racluster and rasort. The -m switch to rasort is simple enough: it specifies what criteria you want the list sorted by. My chosen criteria in this case are twofold: primarily, I want the list sorted by byte-count, and secondarily, by rate. I'll talk more about -m in racluster below. The -s switch does the same thing in both tools. It specifies the output format, which respects the order the arguments are passed on the command line, so the first column is the port number, followed by the byte-count, and finally the rate.

So one day, I opened my mail, and the red-light district report looked like this:

```
39756   1885065707   43076716
61589   1884953405   37598340
```

```
51295   1881932291   19342380
57683   1853690414   19121481
13487   1853657566   19197988
10242   1834900162   18292097
5735    1834731617   17534843
48909   1822365775   19418729
63838   1822191099   19795299
49242   1822184229   19288059
```

"Well that's odd," I thought while logging into the pcap server to verify. Sure enough, my top 10 destinations were all random high-number ports, and the byte counts were huge and yet eerily similar. Nearly two gigabytes per port traversing the router between the staging and dev subnets. My first question was, are these ports being used over and over again, like a virus with a list of ports? Or are each of these unique connections? Let's ask Argus:

```
> ra -r <yesterdays_file> - dst port 39756
  16:25:34.560010  e  D   tcp   10.20.49.21.19026  ->  10.20.33.21.39756
15681122 1885065707   FIN
```

The ra [4] client is the simplest of the Argus clients. It literally stands for "read argus," and its job is to read argus data and output it in human-readable format. The options given you should already recognize. I told it to read from my log file, and gave it a tcpdump-style packet filter with the first destination port on my list. In other words, I took the top destination port and asked ra to return whatever flows used that destination port on that day.

There is only one of them, so that 1885065707 bytes represents a single data connection. One flow to a high-number port looks less like a virus and more like the data channel of an ftp session. Now who would be silly enough to incur *my* wrath by using FTP on the network? Well, this query yields a hint to that question too. They're both database server IPs. A few more ra commands to the listed ports verifies that they're all single connections between the same two Oracle boxes.

Let's see if we can confirm whether this is in fact ftp, and while we're at it, let's see what else these boxes have been saying to each other. I want a list of destination ports that 49.21 has spoken to 33.21 on. Using ra to ask this question would yield every network connection these two hosts have made. So if these boxes had 50 ssh conversations, we would get 50 lines of output for ssh alone. Not what we want. We just want each protocol listed once, so if the boxes used ssh 50 times, we just want ssh mentioned once, kind of like piping the output to sort and then uniq. Argus has an elegant way of doing this in racluster:

```
racluster -r <yesterdays_file> -m proto dport -w - - src host 10.20.49.21 \
and dst host 10.20.33.21  | rasort -m dport -s dport trans bytes  | less
```

The Argus log file can be thought of as a pcap file, with a line for every network conversation. Just reading the file gets you exactly that. But if you want to, you can combine and consolidate these records using whatever metric you want. In this case, we want to combine all the connections that use the same port into a single record. To do this, we use racluster's -m switch. Passing -m proto dport, actually does two things: it first consolidates all of the records that use the same protocol (TCP, UDP, VRRP, etc.) and then further combines all of the records that use the same destination port. When we do this, all of the data we know about those individual records is preserved. For example, the byte counts for each individual ssh connection get added up to a total byte count for all the connections. Argus also keeps a connection counter (called trans) for us, so we know how many connections the record

refers to. We sort the output by destination port number using rasort with -s dport. The output from the above command looks like this:

```
ftp      550        885138
sds        2         21922
5206       2         66124
5367       4     114308574
5509       4      45916440
sgi-es     2        132358
ininme     2         81566
openma     4     151217036
...
```

So these two boxes made 550 ftp control channel connections, which I think verifies our theory about the random high-number ports. Now that you (hopefully) understand racluster's -m switch, go back up and look at the other two reports, which both use -m saddr. Most racluster queries are doing basically the same thing: filtering out some subset of an archive of connections and then combining them based on the metric I'm interested in knowing about. The "chatty kathys" report filters hosts out of a certain subnet and combines them all on source address, so we can see how many combined bytes each source address sent. The "town criers" report does the exact same thing, just with a more specific filter (where the destination host is the network broadcast address).

My last question is, just how much data did these two boxes end up sending to each other? Let's ask Argus:

```
>racluster -r <yesterdays_file> -m daddr - src host 10.20.49.21 \
and dst host 10.20.33.21  -s bytes

363278259124
```

Yeesh. DBAs. There are a few things I want to note about this last command; First, I removed the -w -, because we wanted human-readable output directly from racluster (instead of piping it to rasort). If you forget to do this, you'll get binary gobbledy-gook output and probably hose your term. This is annoying at first, but it's a Pavlov [5] thing; you'll eventually learn to be aware of it. Second, racluster also supports formatted output (-s), as I mentioned above, so if you don't need sorted output, you can dispense with rasort. And last, if I were to back off the search filter and just specify "src host 10.20.49.21", I would get a list of every box 10.20.49.21 had spoken to, suitable for sorting by byte count (do you see why?).

Once you grok racluster's data aggregation concept, every use case immediately becomes kind of obvious. Want to know what Web sites your user base hits the most? Aggregate on destination address (dstaddr) and sort by bytecount (with a filter that excludes internal IPs). Want to know whose sending the virus du jour? Aggregate on destination port (dport). Argus makes getting data like this so easy it's fun. I can (and do) poke around at my Argus data for days (which, in my humble opinion, is a healthy and normal pastime for someone in our profession), but let's face it, eventually you're going to want graphs.

So before you run out and write a Perl script that ties racluster to rrdtool, you should know that the Argus guys already wrote one, and they included it in the client's tarball for you. It's called ragraph [6], works great, and is super easy to use.

Anyway, at this point, I do have a few more questions, but they're not for Argus. If anyone needs me, my clue-by-four and I will be over in the DBA area.

Take it easy.

REFERENCES

[1] PEBCAK: "Problem Exists Between Chair and Keyboard"; http://www.catb.org/jargon/html/P/PEBKAC.html.

[2] ratop: http://www.qosient.com/argus/.

[3] racluster: http://www.qosient.com/argus/.

[4] ra: http://www.qosient.com/argus/.

[5] Pavlov: A man famous for being mean to dogs; http://en.wikipedia.org/wiki/Ivan_Pavlov.

[6] ragraph: http://www.qosient.com/argus/.