DAVE JOSEPHSEN

iVoyeur: pockets-o-packets, part 2

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

SOMEONE ONCE TOLD ME THAT OUR perception of temporal reality changes as we grow older not because our aging brains are slowing down but, rather, because we're gaining context. Things seemed to move so slowly around us when we were young because every new thing we learned made up a huge percentage of what we already knew. When you only knew four people's names, the next new name you learned made up 1/5 of your total knowledge on the subject. How many names do you know now?

I find this a comforting thought, but despite my best efforts I have trouble believing it. The fact that I can't even remember who said it to me is a telling example (I'm almost certain it was a "she"). At any rate, reality seems to me to be speeding up and changing disproportionately to the meager amount of context I've managed to gather (I'm only thirty . . . uh . . . five? six?). But even though I'm barely middle-aged, it seems to me the fundamentals of the world are changing more rapidly than they used to. The incandescent light bulb was invented about 100 years ago. An above-average human life-span to be sure, but it's possible today to have been born in a time when reading by lantern was the norm in North America, and to have lived from that time, through the construction of the grids, air conditioning, the moon landings, nuclear power, solid-state electronics, and the Internet—a mind-boggling influx of reality-shattering changes. Fifty years before that light bulb there were places on the continent for which there were no maps.

For my part, I'm not sure if I'll be able to tell today's children about life before the Internet. I have all sorts of fond memories of how it began—getting my first UNIX shell account from PrimeNet for $6 per month (to MUD), asking my first questions at altavista.digital.com, etc., but I have trouble remembering how I learned things before I could pull my pocket-sized everything-machine out of my pants and ask the magical Google answer hive-mind any question that happened to pop into my head. Before the Internet, how did we research things? I was there, but now I have trouble imagining it. There are several changes that have occurred in my lifetime that I could say this about; before things changed, I didn't know anything was amiss, but afterwards I couldn't imagine what life was like before. I suppose this was what was meant by the phrase “paradigm shift” before the marketeers got hold of it: when things change so fundamentally that the previous reality no longer even seems possible.

Argus

This month I promised I’d talk more about Argus, because I didn’t get nearly as deep as I wanted to with it last time. If you read this column with any regularity, you may have noticed I’m prone to gushing, but there is no amount of gushing I could do over Argus that wouldn’t be deserved. Argus is one of the changes I alluded to above in that I’m not sure how you’d manage a network without it, but I assume it would involve an inordinate amount of guesswork. Argus knows everything about everything that ever transpired on my network. It knows so much about network interaction that it regularly teaches me things about networking in general that I thought I already had a practical understanding of.

I’ve learned so much myself from my interactions with Argus that I recommend it as an aid to people who want to study networking. Not only has Argus taught me things about my network and about networks in general, it has taught me objective truths about the universe. For example, Argus has taught me that I cannot ever, under any circumstances, trust a DBA. I may like them personally, and even respect them professionally, but trust them I simply cannot do, and while this is something I’d often suspected, it is because of Argus that I know it to be the truth.

So let’s get our hands dirty with an examination of our HQ network’s Argus infrastructure, and then I’ll walk through a couple of examples of how I use Argus on a day-to-day basis. Argus is actually two projects: Argus itself [1] is a flow capture daemon, while the Argus Client Programs [2] are the tools that do everything else. The client utilities all have names that begin ‘ra’ (read argus), while the Argus daemon is just called argus. From now on, when I’m talking about Argus the project, to include the client utilities, I’ll use the capital *A*, and when I’m referring to the argus packet capture binary, I’ll use the lowercase *a*.

It can be difficult to understand what all the different little Argus programs do, because there are a lot of them, and there is a lot of functionality overlap between them. argus, for example, really is a packet capture daemon, but it can be used just like one would use tcpdump, to watch packet flows in real time from a given interface. In our design, and in most of the Argus setups I think you’ll likely run into, argus is run on the router, firewall, or device we’re interested in gathering traffic from, and various Argus client programs are used to collect, parse, retransmit, store, and analyze the collected traffic. The Argus documentation refers to argus in this configuration as a “probe.”

Probe Setup

Let’s start with my probe setup. The core routers in our HQ building are OpenBSD boxes on commodity hardware with 10 1 Gbps interfaces each (see Figure 1). Given that they are gigabit devices and have a lot of ports, it would be expensive to use hardware network taps with them, but since they’re multi-purpose OSes on commodity hardware, Argus is a natural fit. Since argus can only listen to a single port at a time, we’ll run one argus instance for each device and then consolidate them into a single data flow before exporting it off the box. It’s possible to run argus as a stand-alone daemon (argus -d), but I prefer to run it under daemontools [3]. My daemontools service directory looks like this:

```
argus.em0 argus.em1 argus.em2 argus.em3 argus.em4 argus.em5 argus.em6
argus.em7 argus.em8 argus.em9 radium
```

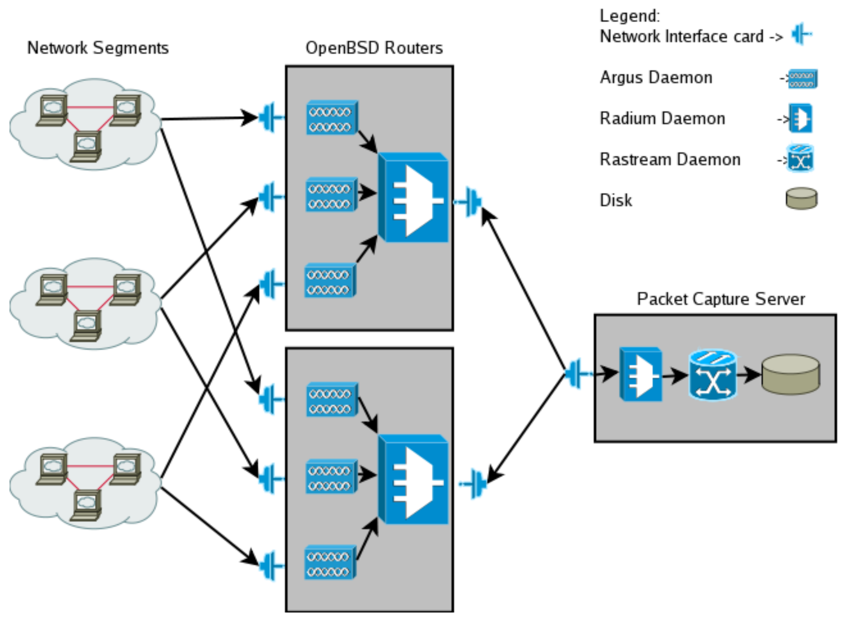

FIGURE 1: HOW ARGUS COMPONENTS FIT TOGETHER IN MY HQ NETWORK ARCHITECTURE.

Each argus.x directory has a daemontools run script, but really these are all just symlinks of the same script, only differing in interface name. Once the symlinks, variables, et al. have been resolved, the argus command for em0 comes out looking like this:

```
/usr/local/sbin/argus -i em0  -B '127.0.0.1' -P "4000" 2>&1
```

And for em1:

```
/usr/local/sbin/argus -i em1  -B '127.0.0.1' -P "4001" 2>&1
```

So you've probably guessed that -i specifies the interface name (just like tcpdump). If I were to stop there, argus would have given human-readable output to STDOUT, but since I specified -P, argus will instead provide Argus-readable data to port 40XX. This is an important concept to understand—every Argus program can give two kinds of output: human-readable and binary Argus data stream. The default is always to output human-readable, but if you want to save output to a file, or pipe the output of one Argus program to another, you need to specify -w. A -w <filename> writes Argus output to a file, while -w - writes it to STDOUT, suitable for piping to other Argus programs. Most Argus programs also support -P, which will write the output to a network socket, and this will always be Argus binary data format. In my daemontools scripts I've also specified the optional -B switch, which tells argus to bind to a specific address rather than 0.0.0.0, which is the default.

So my router has 10 argus daemon processes, each writing to localhost on its own high-number port. Now I want to combine the output from all these daemon processes and provide them as a singular data stream to my syslog/pcap server. To do this I use radium, which you may have noticed among the services directories listed above (radium also supports -d if you want it to run as a stand-alone daemon). radium, which describes itself as a "real-time Argus Record multiplexor," is the first of the Argus clients I'll be talking about, and it's the only Argus client I use on the capture endpoint. radium is intended to do exactly what I need here: it reads argus data from any number of network sockets and outputs a single aggregated data stream to the destination of your choosing. Here's what my radium command looks like when daemontools execs it:

```
radium -S localhost:4000 -S localhost:4001 -S localhost:4002 -S
localhost:4003 -S localhost:4004 -S localhost:4005 -S localhost:4006 -S
localhost:4007 -S localhost:4008 -S localhost:4009 -B 10.20.1.2 -P 3999
```

You've probably guessed that -S specifies a network socket from which to read argus data. You can also read from files with -r, but there are limitations. radium can only read one file at a time, and you cannot read from both sockets and files at the same time. radium, like argus and most of the other Argus tools, supports -B and -P for writing out to sockets and -w for writing out to files. In my environment we provide our aggregated data stream on port 3999 on the router's internal IP, where the syslog/pcap server can read it, and we lock it down with pf locally so only the pcap server can connect.

Data Collector Setup

The /service directory on the pcap server has a radium daemon and a rastream daemon. The radium command on the pcap server looks like this:

```
/usr/local/sbin/radium -S 10.20.1.2:3999 -S 10.20.1.3:3999 -B 127.0.0.1 -P
4001
```

If you are abnormally observant you may have noticed that the internal address of the router ended in ".2". This is because there are actually two routers in a failover setup using CARP. So the pcap server has a radium service that aggregates the flows from each of these routers and provides this stream on localhost:4001. Even if you only have a single probe to read from, I'd recommend using radium to collect it on your pcap box, because this future-proofs things. If/when you get more probes, incorporating them is just an additional -S. rastream's job is to write the data stream to log files, and I could have simply piped radium straight into rastream, but I prefer to write to a socket here instead. This makes it possible for me to come to the pcap box run any client tools against localhost 4001, and get real-time info from all of my probes. radium supports 128 simultaneous client connections, so reading the localhost port on the pcap box doesn't interfere with the logging process. My rastream command looks like this:

```
/usr/local/bin/rastream -S localhost:4001 \
-M time 10m -w "/var/log/argus/%Y-%m-%d/argus.%H.%M.%S" 2>&1
```

The rastream command reads the aggregated probe data from port 4001 and then splits the resulting output into consecutive sections of records based on criteria I give it. These can include file size, time intervals, record count, or even events in the data stream (e.g., rotate the file when you see an SSH connection to a specific host). I use time mode (-M time) with an interval of 10 minutes. I also specify the names of the resultant log files with the -w switch. rastream supports an extended -w option that includes variable expansion with the $. Since I split the logs based on time interval, I use time-based variables, but rastream can actually resolve any printable field in the flow record, which includes byte-counts and the like.

Record Management

Now that the data is on a disk, we can use any of the argus client tools to read it with -r. We can also put some thought into file management and compression. Larger Argus installations can easily generate hundreds of gigs of data per day, so it may not be enough to write the files and forget about them. Let me tell you the three most popular ways to get the data file sizes under control.

The first option you have is a client called racluster. racluster is an amazing Swiss Army knife of a client. In fact, it's the client I use to do most of my data analysis, and I'll be talking more about it in the next issue. With its default options, however, it takes argus input from -S or -r and merges status records from the same flow and argus probe. As argus captures packets from the network, it reports flow statistics every 5 seconds or so. That means there will be *n*/5 records per network connection, where *n* is the number of seconds each connection lasts. Running racluster against an argus dataset merges all of these little records into singular records that represent the entire flow from beginning to end. In my configuration, doing this reduces the size of the data files by 40–60%. We do this via cron once per day. The racluster command looks like this:

```
/usr/local/bin/racluster -R /var/log/argus/${DAY}/ -w /var/log/argus-${DAY}.log
```

racluster is one of several argus clients that support recursively reading directories full of Argus data files with -R. We simply read in a directory of 10-minute-interval files and write out a single file representing the day. racluster needs no other options whatsoever to drastically shrink the size of your archives.

The second option is filtering. Every Argus client, as well as argus itself, supports the use of tcpdump-style traffic filters. Simply give the Argus command as normal, then add a - followed by a tcpdump filter, and you're there. For example, if I wanted to get rid of everything that wasn't TCP traffic, as well as use racluster to shrink my files, I could do:

```
/usr/local/bin/racluster -R /var/log/argus/${DAY}/ -w /var/log/argus-${DAY}.log - tcp
```

The full subset of tcpdump filtering is supported, so things like `- tcp and dst host 192.168.5.1 and dst port 888` work fine. I could also do this earlier in the stream by adding the filter to any of the argus, radium, or rastream commands I've pasted above. Many larger installations run multiple filtered rastream clients against their radium service to store very specific subsets of network traffic in different files, because of storage requirements as well as maintaining separation of duties. This plus racluster is usually enough, but if not, your third option is bzip2. Enough said.

That pretty much covers the Argus infrastructure we're currently using for our office network, which is, I think, a pretty fair example of how end-to-end Argus installations are generally implemented. Unfortunately, I'm out of space this month, so next month, in the third (!) part of this series, I'll cover the really fun stuff: data mining and problem solving with Argus.

Take it easy.

REFERENCES

[1] http://qosient.com/argus/dev/argus-3.0.2.tar.gz.

[2] http://qosient.com/argus/dev/argus-clients-3.0.2.tar.gz.

[3] http://cr.yp.to/daemontools.html.