tcpconnect and tcpaccept for Linux (bcc)

31 Oct 2015

What hosts am I connecting over TCP to?

```
# ./tcpconnect
PID    COMM         IP SADDR            DADDR            DPORT
1479   telnet       4  127.0.0.1        127.0.0.1        23
1469   curl         4  10.201.219.236   54.245.105.25    80
1469   curl         4  10.201.219.236   54.67.101.145    80
11072  ssh          6  ...fe8203ac      ...fe82abcd      22
```

And where am I accepting TCP connections from?

```
# ./tcpaccept
PID    COMM         IP RADDR            LADDR            LPORT
907    sshd         4  192.168.56.1     192.168.56.102   22
907    sshd         4  127.0.0.1        127.0.0.1        22
5389   perl         6  ...fec0ae21      ...fec0ae21      7001
```

Are these tools wrappers to tcpdump? Hell no! I don't want to trace every packet and then filter on SYNs if I don't have to, and I don't. I can just trace the kernel functions doing a TCP socket connect and accept, which are (relative to all packets) less frequent, and so cost lower overhead.

These are two tools I've published in the bcc collection of tools. I wrote about the bcc front end and its eBPF back end previously: in short, eBPF is a new mainline Linux kernel technology that can provide advanced tracing capabilities, and bcc is a python front end to make eBPF easier to use. These tools are pretty new, and rely on Linux 4.1 features.

The current version prints IPv6 addresses as just the last 32 bits as hex. No reason we can't print everything, just haven't coded RFC-4291/RFC-5952 yet.

I first created similar tools using DTrace on Solaris systems at the socket layer (soconnect.d and soaccept.d), and then later using the stable TCP DTrace provider I developed, where they effectively became one-liners. We aren't there yet with bcc/eBPF, but end users can use these tools in the meantime, which come with man pages, examples, and have conventional interfaces with USAGE messages:

```
# ./tcpconnect -h
usage: tcpconnect [-h] [-t] [-p PID]

Trace TCP connects

optional arguments:
  -h, --help         show this help message and exit
  -t, --timestamp    include timestamp on output
  -p PID, --pid PID  trace this PID only

examples:
    ./tcpconnect           # trace all TCP connect()s
    ./tcpconnect -t        # include timestamps
    ./tcpconnect -p 181    # only trace PID 181
```

I previously published a TCP retransmits tool that used ftrace (older built-in Linux tracer) and without kernel debuginfo (an onerous requirement), so that it worked easily on many Linux systems, including our current production ones at Netflix. That tool is in my perf-tools collection. More complicated TCP tools in this fashion are possible, but it gets quite challenging and brittle: tools that are tied to platforms and register usage. I write them as needed to solve problems, but I'm reluctant to share them due to their brittleness.

These bcc tools still have caveats, discussed in the man page (primarily that dynamic tracing is still an unstable interface, and tools may need maintenance between kernel versions), but it's easier to write something that's relatively more stable. I'm looking forward to writing and sharing more TCP (and UDP) tools in the future, and to be on Linux 4.1+ systems that can run these tools. Lots more to do and understand!

You can comment here, but I can't guarantee your comment will remain here forever: I might switch comment systems at some point (eg, if disqus add advertisements).