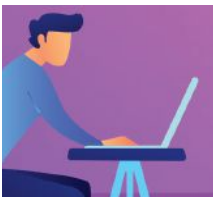


[DIAGNOSING] Excess LDAP Connections Using DTrace by Benedict Reuschling

I manage our CS department's big data cluster and recently faced an interesting issue I was able to diagnose using DTrace. Here is the backstory.



I had set up a couple of student-accessible FreeBSD machines for lab exercises. Students typically log into the cluster machines assigned to them using a combination of SSH and PAM, authenticating themselves against the department's LDAP server. The LDAP server provides all the information the students require, like checking whether a password is correct, what the home directory is, and what shell they're using. For certain systems, students are not allowed access. These nodes have an `AllowedUsers` line in `/etc/ssh/sshd_config` that lists the accounts that are permitted to log in. This setup has worked for a number of semesters without problems, and it is integrated into our installation scripts to let users log into the system from the moment it is installed.

One day, in the middle of the semester, my office phone rang. A colleague from the IT department told me that they had received an unusually high number of requests coming from the big data cluster subnet hitting their two LDAP servers. He gave me a couple IP addresses of the worst offenders that had shown up in their logs. Some were creating multiple requests per second—a number that would typically be much less. Clearly, this was odd and should not have been happening at such a high rate. After hanging up, I concentrated my search on the machines he mentioned. Having a



short list of offending machines was already a good start, as it gave me the material I needed to focus my investigative efforts and also permitted a comparison with systems that exhibit normal behavior.

First, I did a few basic checks: were there any processes running that should not be there, were there too many students working on the machine at the same time, thus creating network requests? Both answers turned out negative, as I was the only person logged into the system. Only the expected lab software was showing up in `top(1)` and `ps(1)` outputs. Next, running `netstat(1)` did not show any unusual connections or open ports other than the ones that were supposed to be open on this host. The same was true for the other hosts in question, so I needed to dig deeper.

I knew the problem somehow involved our LDAP servers and some kind of software running on the hosts connecting to them. But what kind of application was it (third-party or from the OS itself) and how could I get an overview of the number of connections to the LDAP servers for each application? Surely I could run `tcpdump(1)` for a while and see traffic to the LDAP servers go by. It would take more work to get an overview of the initiating applications. I looked at the DTrace one-liners on the FreeBSD wiki [<https://wiki.freebsd.org/DTrace/One-Liners>] and found one that would list the number of connections the local system was making to a remote IP address.

```
sudo dtrace -n 'tcp:::send { @[args[2]->ip_daddr] = count(); }'
```

When I ran it for a couple of seconds and then stopped the trace, I was presented with a two-column output. The first column showed the foreign IP address and the second listed the number of connections made during the trace time in ascending order. That was helpful, as I saw that the two IP addresses of our LDAP servers were showing up among them and there certainly had been a high number of connections during the short while I ran the trace. Over time, these counts increased and the longer I let the DTrace one-liner run, the more probes would fire. Not every program was making a connection every second. Letting the script run for a couple of minutes provided a better overview of what was going on. The only piece of information missing was the actual application making the requests. I extended the script to also give me the executable name (called `execname` in DTrace lingo) like this:

```
sudo dtrace -n 'tcp:::send { @[execname, args[2]->ip_daddr] = count(); }'
```

The output now has three columns: the first has the name of the program or process initiating the connection (for example `sshd` or `sudo`). The second and third are the same as in the previous trace. Since I knew

the target IP address I was looking for (one of our two LDAP servers), I could also filter it more by using

```
sudo dtrace -n 'tcp:::send {@[execname, args[2]->ip_daddr ==  
"IP.address.LDAP.server"]} = count(); }'
```

intr	0	5
sshd	0	23
intr	0	44
postgres	0	63
postgres	0	259
sudo	0	301
intr	0	380
postgres	0	405
intr	0	405
intr	0	422
sudo	0	441
psql	0	475
sshd	0	550
sshd	0	560
sshd	0	665
postgres	0	1073

Looking at the output, a new question arises: why is the ssh daemon sending so much TCP traffic to our LDAP servers? Of course, we're running this script from an SSH session, which means that a good amount of traffic could be generated by us running the script. I stopped the trace and disconnected from the SSH session. After logging in from the server console, I confirmed using `w(1)` that I was still the only user currently logged in. Then I re-ran the trace. To my surprise, `sshd` was still establishing a lot of connections, even though no one was using SSH at the moment.

Compared to other processes on the system, `sshd` was responsible for the second-largest amount of connects, while the leading connection was to the local server IP address. This was fine and did not cause any problems, but why a constant barrage of requests to both LDAP server IPs when no one was actively using SSH sessions? A little further down the list, another thing that caught my eye was the `sudo` process. Of course, I was running my script using `sudo`, so I stopped the trace again and repeated it as the root user. As before, the number of `sudo` connections did not decrease.

Both `ssh` and `sudo` were using the LDAP server—one for enabling certain users to log into the system, while the other gave permissions to run certain local privileged actions. Remember, I had used this configuration for a number of years, but only recently had I received complaints from the IT department about these connections. And since this trace was showing just one machine, you can imagine the number of total requests the LDAP

servers were getting just from this subnet that contains at least 40 nodes with the same configuration (FreeBSD and Linux). Clearly, this needed to be addressed or we'd risk a permanent network ban for those machines, which would be bad news in the middle of the semester when students and researchers were using those machines for their lab work.

Consulting with the IT department, they suggested that I activate `nscd(8)`, the name service caching daemon. The idea was to use the caching service to provide results from the local cache rather than contacting the LDAP servers for each request directly. As the information returned from the LDAP servers would not change very often, a local cache should reduce the load on the servers. It would also increase the local request lookup performance. After reading the respective man pages for `nscd`, `nscd.conf`, and `nsswitch.conf` on FreeBSD, I activated `nscd` using

```
# sysrc nscd_enable=yes
```

Before starting the `nscd` service, I added "cache" statements to `/etc/nsswitch.conf` to make it look like this:


```
group: cache files ldap
hosts: cache files dns
networks: cache files
passwd: cache files ldap
passwd_compat: nis
shells: files
services: compat
services_compat: nis
protocols: files
rpc: files
```

The default `/etc/nscd.conf` has most if not all caches enabled, so there was no need to make any changes to this file. Then I started the `nscd` service using

```
# service nscd start
```

I activated this on the servers that were reported to be the most prominent abusers of the LDAP servers. Then, because other job-related things needed my time, I did not run a second trace to confirm that the requests were lessening in number as a result of activating `nscd`. A week and a half went by without me thinking much about it. There is a lesson here: confirm that an issue has been solved before moving on to other things. Mentally switching back to a problem after a while can be difficult.

In my case, the problem manifested again when I was not at work. On a Wednesday morning, our IT department shut off the network of the entire big data cluster because the number of requests to the LDAP server were so



severe that they were thought to be an internal Denial of Service attack. This caused one of the lab groups that was using the cluster nodes that morning to lose access. Only after the professor attending to that group, who was completely unaware of the network lockout, opened a ticket with the IT department asking for help, did they reestablish the network. The lab group was able to work again; however, the issue remained (and complaints were reaching my inbox, of course). At peak times, a single IP was responsible for 12.5% of total traffic to the LDAP servers. Not good, so a solution had to be found, and quickly.

Apparently, `nscd` had not solved the problem as expected, so I had to get back to the analysis. When searching online for similar issues that other people might have encountered, I did not find anything directly related to my problem. However, I did find discussions about `pam_ldap`, which is the client that I was using on the cluster nodes to establish the connection to the LDAP servers. It turned out that the FreeBSD port had not received any updates since April 1, 2016, and according to some users, the software had been badly designed from the beginning. People recommended an alternative client, written and maintained by Arthur de Jong: `nss-pam-ldapd` [<https://arthurdejong.org/nss-pam-ldapd/>]. Among other things, the port description (`cat pkg-descr` in `/usr/ports/net/nss-pam-ldapd`) listed "less connections to the LDAP server". This sounded exactly like what I was looking for, so I did a test install on one of the cluster nodes that had caused many of those requests.

Not only was the `nss-pam-ldapd` software designed completely from scratch and integrated with a local caching service called `ns1cd`, it was also well documented. Additionally, the software was maintained with the last update of the FreeBSD port in November 2018—the time of the writing of this article. The changes could be implemented without a reboot of the system. A word of warning: errors in the files in `/etc/pam.d` or `/etc/nsswitch.conf` can lock you out of the system, even the root user. When that happens, you need to repair this in single-user mode or using a live CD.

Once the changes had been put in place and the `ns1cd` service had been started, I ran the DTrace script again. I found an entry for `ns1cd` in the first column connecting to the LDAP server (I had disabled `nscd` earlier as it did not help). This was now the caching daemon making the calls, providing the results to the local services (`sudo` and `ssh`) asking for the information. The number of connections to the LDAP servers was greatly reduced in the DTrace output. On the server side, our IT department also confirmed that the traffic had gone down to normal levels again. Needless to say, I rolled out the `nss-pam-ldapd` service with `ns1cd` on all the other machines. I also updated our install scripts to use `nss-pam-ldapd` instead of `pam-ldap` when a system is installed. Note there is also a version available with SASL support called `nss-pam-ldapd-sasl`.

The nice thing about this exercise was that I did not have to think much about writing the DTrace probes myself. I could simply access them from the library of handy DTrace one-liners on the FreeBSD wiki [https://wiki.freebsd.org/DTrace/One-Liners]. There are already a couple of good and ready-to-use one-liners there for various areas like storage, networking, and syscalls. With a bit of modification, it is fairly straightforward to construct a probe that provides the information you are looking for to help diagnose the more difficult problems. ●



Benedict Reuschling joined the FreeBSD Project in 2009. After receiving his full documentation commit bit in 2010, he actively began mentoring other people to become FreeBSD committers. He joined the FreeBSD Foundation in 2015, where he is currently serving as vice president. Benedict has a Master of Science degree in Computer Science and is teaching a UNIX for software developers class at the University of Applied Sciences, Darmstadt, Germany. Together with Allan Jude, he is host of the weekly BSDNow.tv (<http://BSDNow.tv>) podcast.

ZFS experts make their servers

Now you can too. Get a copy of.....

ZING

Choose ebook, print, or combo. You'll learn to:

- Use boot environment, make the riskiest sysadmin tasks boring.
- Delegate filesystem privileges to users.
- Containerize ZFS datasets with jails.
- Quickly and efficiently replicate data between machines.
- Split layers off of mirrors.
- Optimize ZFS block storage.
- Handle large storage arrays.
- Select caching strategies to improve performance.
- Manage next-generation storage hardware.
- Identify and remove bottlenecks.
- Build screaming fast database storage.
- Dive deep into pools, metaslabs, and more!



WHETHER YOU MANAGE A SINGLE SMALL SERVER OR INTERNATIONAL DATA CENTERS, SIMPLIFY YOUR STORAGE WITH

Link to:

<http://zfsbook.com>

FREEBSD MASTERY: ADVANCED ZFS. Get it Today!