Anatomy of a Linux DNS Lookup – Part I

6 Minutes

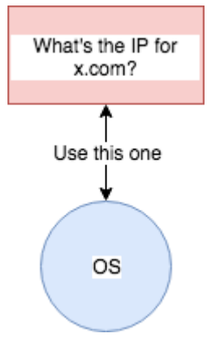
Since I work (https://zwischenzugs.com/2017/10/31/a-complete-chef-infrastructure-on-your-laptop/) a (https://zwischenzugs.com/2017/03/04/a-complete-openshift-cluster-on-vagrant-step-by-step/) lot (https://zwischenzugs.com/2017/03/04/migrating-an-openshift-etcd-cluster/) with (https://zwischenzugs.com/2017/03/04/1-minute-multi-node-vm-setup/) clustered (https://zwischenzugs.com/2017/03/18/clustered-vm-testing-how-to/) VMs (https://zwischenzugs.com/2017/10/27/ten-things-i-wish-id-known-before-using-vagrant/), I've ended up spending a lot of time trying to figure out how DNS lookups (https://zwischenzugs.com/2017/10/21/openshift-3-6-dns-in-pictures/) work. I applied 'fixes' to my problems from StackOverflow without really understanding why they work (or don't work) for some time.

Eventually I got fed up with this and decided to figure out how it all hangs together. I couldn't find a complete guide for this anywhere online, and talking to colleagues they didn't know of any (or really what happens in detail)

So I'm writing the guide myself.

Turns out there's quite a bit in the phrase 'Linux does a DNS lookup'…

Other posts in the series:



"How hard can it be?"

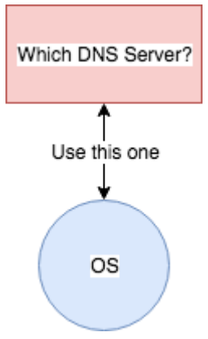
These posts are intended to break down how a program decides how it gets an IP address on a Linux host, and the components that can get involved. Without understanding how these pieces fit together, debugging and fixing problems with (for example) `dnsmasq`, `vagrant landrush`, or `resolvconf` can be utterly bewildering.

It's also a valuable illustration of how something so simple can get so very complex over time. I've looked at over a dozen different technologies and their archaeologies so far while trying to grok what's going on.

I even wrote some automation code (https://github.com/ianmiell/shutit-linux-dns/blob/master/linux_dns.py) to allow me to experiment in a VM. Contributions/corrections are welcome.

Note that this is not a post on 'how DNS works'. This is about everything up to the call to the actual DNS server that's configured on a linux host (assuming it even calls a DNS server – as you'll see, it need not), and how it might find out which one to go to, or how it gets the IP some other way.

1) There is no such thing as a 'DNS Lookup' call



This is NOT how it works

The first thing to grasp is that there is no single method of getting a DNS lookup done on Linux. It's not a core system call with a clean interface.

There is, however, a standard C library call called which many programs use: `getaddrinfo (http://man7.org/linux/man-pages/man3/getaddrinfo.3.html)` . But not all applications use this!

Let's just take two simple standard programs: `ping` and `host` :

```
root@linuxdns1:~# ping -c1 bbc.co.uk | head -1
PING bbc.co.uk (151.101.192.81) 56(84) bytes of data.
```

```
root@linuxdns1:~# host bbc.co.uk | head -1
bbc.co.uk has address 151.101.192.81
```

They both get the same result, so they must be doing the same thing, right?

Wrong.

Here's the files that `ping` looks at on my host that are relevant to DNS:

```
root@linuxdns1:~# strace -e trace=open -f ping -c1 google.com
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
open("/lib/x86_64-linux-gnu/libcap.so.2", O_RDONLY|O_CLOEXEC) = 3
open("/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
open("/etc/resolv.conf", O_RDONLY|O_CLOEXEC) = 4
open("/etc/resolv.conf", O_RDONLY|O_CLOEXEC) = 4
open("/etc/nsswitch.conf", O_RDONLY|O_CLOEXEC) = 4
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 4
open("/lib/x86_64-linux-gnu/libnss_files.so.2", O_RDONLY|O_CLOEXEC) = 4
open("/etc/host.conf", O_RDONLY|O_CLOEXEC) = 4
open("/etc/hosts", O_RDONLY|O_CLOEXEC)  = 4
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 4
open("/lib/x86_64-linux-gnu/libnss_dns.so.2", O_RDONLY|O_CLOEXEC) = 4
open("/lib/x86_64-linux-gnu/libresolv.so.2", O_RDONLY|O_CLOEXEC) = 4
PING google.com (216.58.204.46) 56(84) bytes of data.
open("/etc/hosts", O_RDONLY|O_CLOEXEC)  = 4
64 bytes from lhr25s12-in-f14.1e100.net (216.58.204.46): icmp_seq=1 ttl=63 time=13.0 ms
[...]
```

and the same for `host`:

```
$ strace -e trace=open -f host google.com
[...]
[pid  9869] open("/usr/share/locale/en_US.UTF-8/LC_MESSAGES/libdst.cat", O_RDONLY) = -1 ENOENT (No such file or directory)
[pid  9869] open("/usr/share/locale/en/libdst.cat", O_RDONLY) = -1 ENOENT (No such file or directory)
[pid  9869] open("/usr/share/locale/en/LC_MESSAGES/libdst.cat", O_RDONLY) = -1 ENOENT (No such file or directory)
[pid  9869] open("/usr/lib/ssl/openssl.cnf", O_RDONLY) = 6
[pid  9869] open("/usr/lib/x86_64-linux-gnu/openssl-1.0.0/engines/libgost.so", O_RDONLY|O_CLOEXEC) = 6[pid  9869] open("/etc/resolv.conf", O_RDONLY) = 6
google.com has address 216.58.204.46
[...]
```

You can see that while my `ping` looks at `nsswitch.conf`, `host` does not. And they both look at `/etc/resolv.conf`.

We're going to take these two `.conf` files in turn.

2) NSSwitch, and `/etc/nsswitch.conf`

We've established that applications can do what they like when they decide which DNS server to go to. Many apps (like `ping`) above can refer (depending on the implementation **(*)**) to NSSwitch via its config file `/etc/nsswitch.conf`.

(*) There's a surprising degree of variation in ping implementations. That's a rabbit-hole I *didn't* want to get lost in.

NSSwitch is not just for DNS lookups. It's also used for passwords and user lookup information (for example).

NSSwitch was originally created as part of the Solaris OS to allow applications to not have to hard-code which file or service they look these things up on, but defer them to this other configurable centralised place they didn't have to worry about.

Here's my `nsswitch.conf`:

```
passwd:         compat
group:          compat
shadow:         compat
gshadow:        files
hosts: files dns myhostname
networks:       files
protocols:      db files
services:       db files
ethers:         db files
rpc:            db files
netgroup:       nis
```

The 'hosts' line is the one we're interested in. We've shown that `ping` cares about `nsswitch.conf` so let's fiddle with it and see how we can mess with `ping`.

- Set `nsswitch.conf` to only look at 'files'

If you set the `hosts` line in `nsswitch.conf` to be 'just' `files`:

```
hosts: files
```

Then a `ping` to google.com will now fail:

```
$ ping -c1 google.com
ping: unknown host google.com
```

but `localhost` still works:

```
$ ping -c1 localhost
PING localhost (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.039 ms
```

and using `host` still works fine:

```
$ host google.com
google.com has address 216.58.206.110
```

since, as we saw, it doesn't care about `nsswitch.conf`

◦ Set `nsswitch.conf` to only look at 'dns'

If you set the `hosts` line in `nsswitch.conf` to be 'just' dns:

```
hosts: dns
```

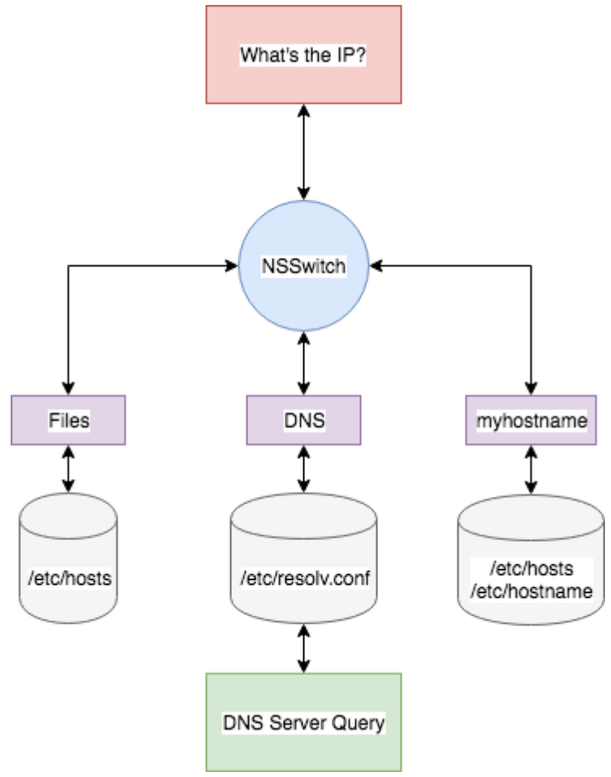
Then a `ping` to google.com will now succeed again:

```
$ ping -c1 google.com
PING google.com (216.58.198.174) 56(84) bytes of data.
64 bytes from lhr25s10-in-f174.1e100.net (216.58.198.174): icmp_seq=1 ttl=63 time=8.01 ms
```

But `localhost` is not found this time:

```
$ ping -c1 localhost
ping: unknown host localhost
```

Here's a diagram of what's going on with NSSwitch by default wrt `hosts` lookup:



My default ' ***`hosts:`*** *' configuration in* ***`nsswitch.conf`***

3) /etc/resolv.conf

We've seen now that `host` and `ping` both look at this `/etc/resolv.conf` file.

Here's what my `/etc/resolv.conf` looks like:

```
# Dynamic resolv.conf(5) file for glibc resolver(3) generated by resolvconf(8)
#     DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRITTEN
nameserver 10.0.2.3
```

Ignore the first two lines – we'll come back to those (they are significant, but you're not ready for that ball of wool yet).

The `nameserver` lines specify the DNS servers to look up the host for.

If you hash out that line:

```
#nameserver 10.0.2.3
```

and run:

```
$ ping -c1 google.com
ping: unknown host google.com
```

it fails, because there's no nameserver to go to (*).

* Another rabbit hole: `host` appears to fall back to 127.0.0.1:53 if there's no nameserver specified.

This file takes other options too. For example, if you add this line to the `resolv.conf` file:

```
search com
```

and then `ping google` (sic)

```
$ ping google
PING google.com (216.58.204.14) 56(84) bytes of data.
```

it will try the `.com` domain automatically for you.

End of Part I

That's the end of Part I. The next part will start by looking at how that resolv.conf gets created and updated.

Here's what you covered above:

- There's no 'DNS lookup' call in the OS
- Different programs figure out the IP of an address in different ways
  - For example, `ping` uses `nsswitch`, which in turn uses (or can use) `/etc/hosts`, `/etc/resolv.conf` and its own hostname to get the result
- `/etc/resolv.conf` helps decide:
  - which addresses get called
  - which DNS server to look up

If you thought that was complicated, buckle up…

Published by zwischenzugs

View all posts by zwischenzugs

25 thoughts on “Anatomy of a Linux DNS Lookup – Part I”

Pingback: Recommended Read: Anatomy of a Linux DNS Lookup – Part I | thechrisshort
Gordon Messmer says:
June 9, 2018 at 5:08 pm
This is a good write-up. I think there are a couple of things you could add, because the POSIX DNS API is actually a little more complex than this, even.

(First, to pick a nit: What you’re describing has nothing to do with Linux. The specification is POSIX, and the implementation is GNU. All of this applies equally to GNU/Windows, aka WSL or “bash on Windows”, where there is no Linux component.)

GNU libc provides three different name resolution APIs. There is the low-level DNS resolver library (RESOLVER(3)) which implements a BSD specification, there is gethostbyname (GETHOSTBYNAME(3)) and its related functions which implement an obsolete POSIX specification, and there is getaddrinfo (GETADDRINFO(3)) which is the modern POSIX API for name resolution.

Applications fall into one of maybe three categories. Maybe two. It’s subjective. The first is older applications which still use the obsolete gethostbyname() API. The second is newer applications that use getaddrinfo(). Both of those will go through the GNU libc NSS service, so they’ll parse /etc/nsswitch.conf, and from there they might use /etc/hosts, /etc/resolv.conf, /etc/hostname, and they might also use multicast DNS or an NSS caching service like nscd or sssd. The getaddrinfo API will also consult /etc/gai.conf to order address results according to system preference.

The third group of applications (or second group, depending on your point of view) is applications that don’t use the POSIX NSS APIs at all. This includes the “host” utility, because “host” is actually one of the applications included with ISC BIND, and it is intended specifically to interface with DNS directly, and not the system NSS API. It also includes applications like Firefox which bypass NSS and implement their own DNS API for performance reasons. And it includes applications which need access to records other than A and AAAA records. Such applications will need to use the resolver library or their own DNS client library to access MX or TXT records.

I also want to note that “ping localhost” should not fail when nsswitch refers only to DNS. A DNS server is required by RFC to resolve the name “localhost”. The DNS service that you’re using is not compliant with standards, due to a configuration error.

And, finally, that the documentation for gethostbyname indicates that it will fall back to a DNS server on 127.0.0.1, just like ISC’s “host” application. The documentation for getaddrinfo does not contain such a note, but I suspect that it will do the same since those are both part of GNU libc’s NSS API.

Thanks for the write-up. I hope this helps.

↳ Reply
zwischenzugs says:
June 10, 2018 at 1:32 pm
This ^^^ is absolute gold, thank you so much for posting it. I’ll work it in as I go. Some of it I knew but didn’t want to overwhelm the reader, but a lot of it I did not know at all.

↳ Reply
Pingback: Anatomy of a Linux DNS Lookup – Part II – zwischenzugs
Khurram Subhani says:
June 19, 2018 at 2:52 pm
maybe a silly question to ask: How does nsswitch.conf know what files or dns is/pointing to? Is there a config file which points ‘files’ to’/etc/hosts’. The reason why I am asking is that I saw above ‘myhostname’ as an option which points to /etc/hosts & /etc/hostnames

↳ Reply

zwischenzugs says:
June 19, 2018 at 3:08 pm
It's not a silly question, it's just the kind of question I need to ask and answer to write these things.

↳ Reply

zwischenzugs says:
June 19, 2018 at 4:01 pm
The man nsswitch.conf page suggests they're hard-coded. You could get the source and examine it to be sure…

↳ Reply

Uwe says:
June 22, 2018 at 8:25 pm
that's exactly the reason why I always use getent hosts example.org when shell scripting. it uses the c call, queries the system's host database and honors nsswitch. the rabbit hole goes even deeper .I've seen a dozen of different output formats from host – a nightmare to parse but I won't go into details

↳ Reply

Pingback: Anatomy of a Linux DNS Lookup – Part III – zwischenzugs
Pingback: DNS lookup tutorial (1) | 0ddn1x: tricks with *nix
Pingback: Links 10/7/2018: Wine 3.12, FreeNAS 11.2 Beta, GNU Helps Journalism | Techrights
Pingback: Weekly DevOps News #1 - Codeogre
Pingback: Anatomy of a Linux DNS Lookup – Part IV – zwischenzugs
Pingback: Anatomy of a Linux DNS Lookup – Part V – Two Debug Nightmares – zwischenzugs
Pingback: Bug Bytes #23 – 20K IDOR Trick, Bug Bounty Vloggers everywhere & Persistent Burp Collaborator – INTIGRITI

andhe says:
December 1, 2019 at 5:15 pm
You kind of already toughed on it with the host utility, but for extra points on how screwed up things are you could look at things that does not use the C standard library (libc). An "interesting" read is diving into how golang applications resolves things. The golang library tries to guess things about nss and fake it. A testcase to show how broken it is can for example be to install nss-myhostname and rm /etc/hosts … The nss module should allow you to resolve localhost and your hostname, but since golang lib doesn't actually call into the libc and thus nss stack it doesn't work. Run tcpdump and you'll see the queries go to your (isp) dns resolver!

↳ Reply

Pingback: How To Flush DNS Cache on Linux – devconnected - DzTechno - DZTECHNO!
Pingback: How To Flush DNS Cache On Linux - RSSFeedsCloud
Pingback: New top story on Hacker News: Anatomy of a Linux DNS Lookup - Welcome to world wide tech news
Pingback: Futureseek Daily Link Review; 5 June 2021 | Futureseek Link Digest

vinci says:
June 7, 2021 at 4:52 pm
I have one question: what Linux did you test this on?

↳ Reply

zwischenzugs says:
July 12, 2021 at 9:52 am
Ubuntu, mostly.

↳ Reply

Pingback: How To Flush DNS Cache on Linux – devconnected - DZTECHNO
Pingback: How does Curl resolve DNS in linux? – CodePT
Pingback: Membersihkan Cache DNS Linux - Komunitas Klub Cahaya