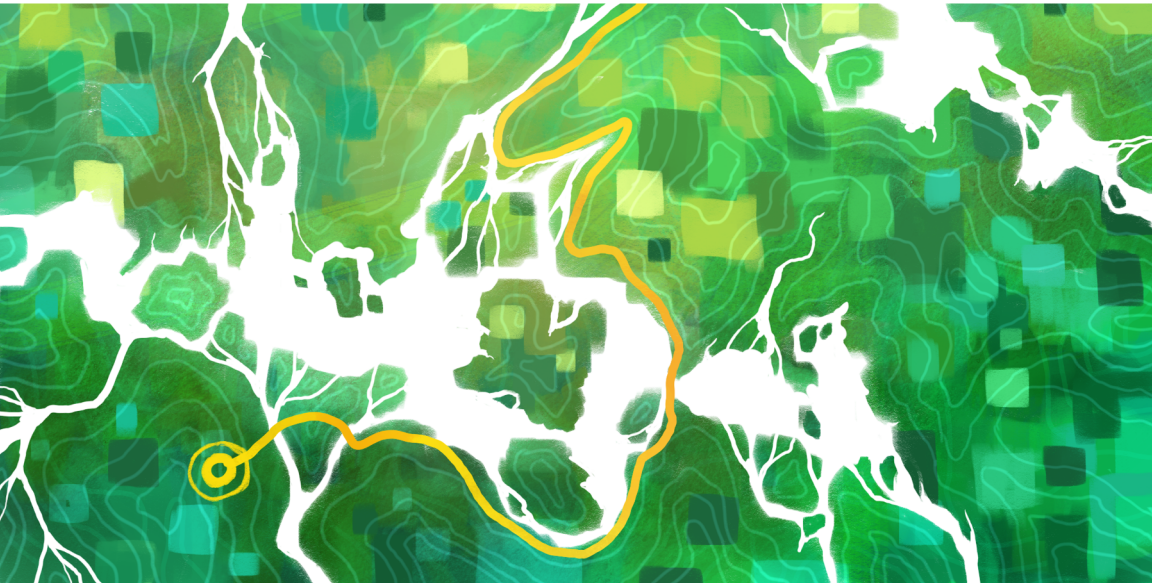
Domain Name Sanity

by Edward Loveall

thoughtbot



Domain Name Sanity

Edward Loveall

August 2, 2016

Contents

This book is dedicated to my wife, Elizabeth, who is super bummed I finished writing a book before she did.

Introduction

Welcome

I didn't start out knowing much about DNS. Sure, I knew there was something called an "A record" and a "CNAME." I knew it took "up to 48 hours for changes to propagate across the web." I knew that GoDaddy was "terrible." But when faced with a placeholder page that said "website coming soon," I felt like a donkey at a computer: not sure where to start.

I'm a web developer most days, so stepping through a problem methodically is my preferred way of working. In DNS, as in development, problems inevitably arise. Unlike development, however, DNS appears to defy a methodical approach to solving these problems. The tools and technologies are scattered and often don't resemble each other at first glance. We will walk through these tools together so that we can understand more or less how they work, and fix problems as they appear.

TLS has its own set of quirks, too. It is becoming more and more popular, and in some cases required, to have a secure website. This is easier said than done, however. The process involved in procuring and installing a certificate properly can be daunting to the uninitiated. Don't worry, though–we'll look at it in detail.

Finally, because I am a web developer, this book is very website-centric, but all techniques should work fine for a server with no outward-facing website, such as an API or an FTP server.

By the time we're done, you will be able to set up eight different kinds of records, use six command line tools, deal with a handful of common problems, and make

a website secure.

Let's get started!

Who is this book for?

As far as the subject matter for this book goes, I hope to make this whole process a bit more enjoyable than your average tech book by using lame jokes and silly examples. Frankly, writing in lame jokes is probably the only way I could ever finish writing this book.

Broadly, we are going to talk about DNS, how it works, and how to troubleshoot it. We are also going to look at TLS certificates and walk through not only how they work, but how to get and install one. If this sounds useful, then welcome.

As far as what you know coming into this book, I imagine you as a developer, or at a least technically minded individual, are probably a bit curious about how DNS works behind the scenes and have used a command line at least once in your life.

Who is this book not for?

Have you *never* touched the command line before and are totally scared of it? Do you have *no* interest in having your own website? Then this book is probably not for you. No harm done, we'll give you a refund. I'd still urge you to give it a shot, though. Domains, DNS, and TLS can be very useful things to know about.

How to Read This Book

Considering you are reading with your eyes, this might be a confusing section to come across. I'll assume you are literate, but perhaps not familiar with my particular conventions.

New terminology

I will try to define new terms as I go. There is, however, a good chance that I may assume (wrongly, perhaps) a term is well known enough that I don't need to explain it on the spot. I might also accidentally forget to define something. If you find yourself confused and needing a quick reminder or introduction to a term, check the glossary where, hopefully, I have defined all the technical terms that you're curious about.

Literal text

In the case where I'm trying not to use words to describe things, but to show something exact, I will specify it in a monospaced font `like this`. You'll see this scattered throughout the book where there is a small piece of text (i.e., one line or less) that I expect you to type into a field or configuration file.

When we need to use a command line tool like `dig`, it's often useful to show an example of the command and its response. In those situations, I'll be using code block formatting, like so:

```
$ some command

some response

...
```

I also use this formatting when I'm showing multiple lines of text, like in a configuration file.

Truncation

Three dots `...` will represent some kind of truncated response. It's not that I couldn't copy and paste something, but seeing a large block of what looks like gibberish is draining to read. If something is important, don't worry, I'll include it.

Shell commands

Also, you may have noticed the dollar sign at the start of the command. This represents a command I expect you to copy and paste into your shell if you want to follow along and see a similar response. They also serve to separate commands from responses.

Note that the dollar sign is *not* included in the command. It's what we call the "prompt." You'll probably see something like your computer name, current folder, and/or username, followed by the dollar sign in your prompt. For brevity, we can just shorten to the single symbol.

So in a case like this:

```
$ hello
```

I just expect you to type the five letters "hello", not "$ hello".

Short `dig` responses

I also use the command line tool `dig` liberally throughout the book as it's extremely useful for inspecting existing domains. It has a relatively verbose response by default. To try and reign in these responses, I use the `+short` option on most `dig` commands:

```
$ dig +short example.com
```

```
104.131.191.2
```

Note that you most definitely can leave off the `+short` part if you want the full, lengthy response from `dig`.

The example domain

Inevitably, we're going to talk about domains in this book about the domain name system. Instead of trying to stay abstract, I've picked a domain we'll use in examples: donkeyrentals.com.

Why donkey rentals? Mostly it makes the book a bit more fun to read. I couldn't type `example.com` for chapter after chapter and keep my sanity.

It's also a real domain that works. As much as possible I've tried to make all the example `dig` queries the same as they are in the book so we can see them in action. Feel free to query the domain as much as you like to try and get the hang of any of the concepts in the book. It's there for you, dear reader!

Software

You will need a few pieces of software to follow along with this book. The appendix at the back of the book should be able to help you install anything missing. The software you will need is as follows:

- A web browser
- A command line (like the Command Prompt on Windows or the Terminal on Mac OS X)

On the command line, these pieces of software should be installed:

- dig
- openssl (or equivalent)
- whois

These are optional, but could prove useful anyway:

- host
- nslookup
- ping

If you're not sure if you have these pieces of software, the appendix contains instructions to check each one and install it if it is missing.

Registering A New Domain

Registering a domain isn't the simplest process. Which, like, duh, that's probably why you bought this book. But it doesn't have to be rocket science either. We need to figure out if our domain is taken, find a registrar (a company that can sell us a domain), buy the domain, and, finally, fill out our WHOIS information. Okay, it does sound a little complicated, so let's walk through each of the steps.

Step One – Figure Out if the Domain Is Already Taken

Let's say we want the domain `donkeyrentals.com`. To register that name, we need to know if it's already taken. We could go visit the domain in our web browser and see if a website appears, but even if there's no website, that doesn't necessarily mean it's not registered. Another way is to go to a domain registrar, type in the domain, and see if it's taken. There's a catch, though: *domain name front running*. This happens when a company registers a domain as soon as someone has searched for it. Some registrars have been caught doing this.

To make sure that doesn't happen, skip the registrar altogether by using the command line utility `whois`:

```
$ whois donkeyrentals.com
```

`whois` (pronounced "who is") retrieves information about who has rights to a domain name. Notice I didn't mention ownership of a domain name. No one ever actually owns a domain. We're just leasing it for a certain amount of time. To confuse things even further, the protocol used by whois to get its information is

also called WHOIS, but with uppercase letters. In summary: we can use `whois` to retrieve WHOIS information. Makes sense, right?

Using the command line tool isn't the only way. We can also use Internic's WHOIS service, a site that continues to hold strong to the design aesthetic of the early 1990s. The only caveat of the service is it only supports certain domains, mostly the more common ones such as `.com`, `.net`, and `.org`.

From now on, I'm not going to use the period for common domains since it's hard to read and isn't technically accurate, so you'll see `com`, `net`, etc. instead.

If `donkeyrentals.com` is taken, we need to look for a different domain, so let's try `donkeyrentals.dentist` instead. If we're trying to retrieve WHOIS information for a domain that Internic doesn't support, we have to use a special WHOIS server. So we go to the Root Zone Database, find the top-level domain we're looking for, and see if they have a WHOIS server. For this step, I looked up the whois server for `dentist` domains and found `whois.rightside.co`. We can either visit that site to look up our domain, or we can use the `whois` tool with the `-h` option:

```
$ whois -h whois.rightside.co donkeyrentals.dentist
```

However, using either method returns a bunch of text. Because there is no standard for what WHOIS returns, this could be any text. We're looking for text that, in so many words, says *We have no record of this domain so therefore it's open for registration.* Common ways to phrase this are: `No match for "EXAMPLE.COM"` or `No entries found`.

We'll fill out our own WHOIS information later on.

Step Two – Find a Registrar

Now that we know our domain is available, we need to lease it from a registrar. Picking a registrar is mostly a personal preference, so here's what I look for:

Wide selection of top-level domains

With so many top-level domains (TLDs) like `com`, `io`, `co`, and `equipment`, available, we have to make sure our registrar can handle all the domains we might want to register. (If you're looking for business ideas, `donkeyrentals.equipment` sounds quite promising. But I'm just the idea guy, so you'll have to run with it.) Whichever

domains we want to register, we need to be able to manage them all from one spot if possible.

A nice control panel

The control panel is where we will spend the most time with a registrar, so it's worth researching. This is super important, but harder to know beforehand if the control panel is good before you sign up with that registrar. Most sites will have a nice landing page, but some may be hiding a crap control behind it. Ideally, we can find screenshots of the control panel or use a demo/trial. There's probably no *perfect* registrar out there, but if it looks like the registrar has never heard of a web designer, we may have to look elsewhere.

Other features

Many registrars offer features beyond domain name administration. Some offer email, web hosting, or DNS management (which is technically different from domain registration). Some offer one-click configuration to set up your domain with your favorite services. There are even services that just *give* you a domain for free when you purchase their other services. Others will totally scam you (this is not usually a desired feature).

Most of these options are personal preferences, but none of them are critical. It's hard to know exactly what you'll need in the future, but this gives you an idea of what to look for as you research.

If you're a bit overwhelmed now and just want a suggestion, here are some of my favorites.

Step Three – Lease the Domain

The price for a single domain ranges from free all the way up to $100+/year. Some registrars charge monthly for other niceties like WHOIS privacy, email, or website hosting. There's no standard, so they can charge (or not charge) whatever they want. There are some commonalities among registrars, however. Long-existing domains like `com`, `net`, and `org` are usually pretty cheap–around $10–while newer or specialized domains such as `luxury` and `loans` generally cost more, just like their real-world counterparts.

Step Four – Complete WHOIS Information

After signing up for an account, entering the credit card info, and agreeing to a terms of service we probably should have read but didn't, we need to enter WHOIS information, including, among other things, a phone number, email, and physical address for administrator, billing, and technical contacts.

Privacy vs legal concerns

We may not want to put sensitive information on the internet, which is understandable. We could use non-personal email addresses, PO boxes, and fake phone numbers, but those sound like a lot of work for a simple domain name. Many registrars offer a service called WHOIS Privacy that lists obscuring information such as *their* contact info instead. This can be a nice alternative, but the same info can also be used to determine who has rights to a domain, so if our info isn't there, it can be harder to prove that we own it.

In one case, a registry shut down because its owner spent thousands of dollars of customer money on liposuction, Escalades, and a Miami Penthouse. As ridiculous as this sounds, customers who had WHOIS Privacy enabled couldn't prove that they had rights to their domains as the site slowly imploded. This is a worst-case scenario, but it's important to be aware of the potential tradeoffs of WHOIS privacy.

Next Steps

Now that we have a domain to play with, we can start talking about all the other configuration options available to us, such as SSL certificates, email, and subdomains. Right now, the most important task we should be focused on is pointing our domain to our content or servers. That is done with DNS records, which is exactly what the next chapter is all about.

Types of DNS Records

A

A records are the bread and butter of DNS, and probably the least complex. A records always (and only) point toward an IPv4 address such as `104.131.191.2`. A good way to remember this is "A for Address."

Here's an example of how we could configure an A record:

- **Hostname**: `@`
- **Record Type**: `A`
- **IP Address**: `104.131.191.2`

The `@` symbol means **apex domain** and has nothing to do with the "at sign" in an email address. It's also called the bare, top, root, or naked domain. It's a fancy way to refer to the domain without any label in front of it, i.e., `donkeyrentals.com` rather than `www.donkeyrentals.com`. Sometimes you specify the apex domain by putting nothing for the hostname, but I'll use `@` throughout this book.

Once that record is set up, we can check our work by using `dig`:

```
$ dig +short donkeyrentals.com A

104.131.191.2
```

We've made our domain name point to essentially the same thing as the IP address. In practice, this only works one way. Since there might be many websites

hosted at that IP address, we aren't guaranteed to get to *our* website when we visit it. Our server at `104.131.191.2` knows it's getting a request for `donkeyrentals.com` but it must be configured to handle that *specific* domain, not just the IP address.

We can also use A records with subdomains:

- **Hostname**: `accessories`
- **Record Type**: `A`
- **IP Address**: `104.131.191.2`

This will set up a similar record as above, but for the subdomain `accessories.donkeyrentals.com`:

```
$ dig +short accessories.donkeyrentals.com A

104.131.191.2
```

Once both A records are set up, we will have an example of more than one website hosted at the same IP address. Our main corporate enterprise's website is `donkeyrentals.com`. For things like diamond-studded donkeyshoes, we have our accessories site, which is at `accessories.donkeyrentals.com`. We host both websites at `104.131.191.2`, but each have different content.

AAAA

These records are the same as A records, but with one exception: they point to an IPv6 address instead of an IPv4. IPv6 addresses look something like this: `2620:0000:0861:ed1a:0000:0000:0000:0001`. That's eight sets of four hexadecimal digits, which is way too long. However, they can be severely shortened: `2620:0:861:ed1a::1`.

Here's how to shorten your very own IPv6 address:

- Drop the leading zeros in each section: `2620:0:861:ed1a:0:0:0:1`
- Replace consecutive groups of all zeros with two colons: `2620:0:861:ed1a::1`

Also:

- If there are more than one group of all zeros, replace the longest group.
- If other groups are the same length, replace the left-most group.

Other than being longer and harder to type, they serve the same function as IPv4: to be an address for our server. Technically, these are pronounced as "*quad-A*" records, but I always scream "*AAAAAHHHH!!!!*" in my head.

Here's an example configuration:

- **Hostname**: `@`
- **Record Type**: `AAAA`
- **IP Address**: `2620:0:861:ed1a::1`

```
$ dig +short donkeyrentals.com AAAA

2620:0:861:ed1a::1
```

As the internet runs out of IPv4 addresses, IPv6 is becoming more important. Soon you might not even be able to use an IPv4 address.

If you want to visit these addresses in your browser, surround them with brackets: `http://[2620:0:861:ed1a::1]`. This should work for any length of IPv6 address. In practice I found that these urls are often blocked or not recognized, so your mileage my vary. If they don't work for you, ask around to see if someone can try the address for you.

CNAME

You need to know going into this section that CNAMEs are odd and hard to understand. They have their uses, which we'll get into below, but don't worry if at first read you are as confused as I was when I began learning about CNAMEs. So saddle up on your rental donkey and let's get moving.

CNAME records are the other side of the coin from A and AAAA records. Instead of pointing toward an IP address, they can only point toward another domain. Think of CNAMEs as making one domain accessible from another domain. For example, a common configuration is to point `www` toward our apex domain so both addresses display the same website:

- **Hostname**: `www`
- **Record Type**: `CNAME`
- **Target Host**: `donkeyrentals.com`

```
$ dig +short www.donkeyrentals.com CNAME

donkeyrentals.com.
```

Now `www.donkeyrentals.com` will point to `donkeyrentals.com`.

That's not all. You can also point a CNAME record at any domain, even one that isn't ours:

- **Hostname**: `redirect`
- **Record Type**: `CNAME`
- **Target Host**: `rabbitrentals.com`

```
$ dig +short redirect.donkeyrentals.com CNAME

rabbitrentals.com.
```

Now, when people try to visit `redirect.donkeyrentals.com`, they get *hopped* over (see what I did there?) to our competitor's site.

When to Use CNAMEs

Why would we use a CNAME record instead of an A? Well, for the most part, you wouldn't. A records are preferred because they are much more direct. When your browser sees an A record, it gets an IP address. That's it. Requesting a CNAME record is different. It looks up the record, sees that it's a CNAME, looks at what the record is pointing toward, and then *restarts* the request. This will make requests take longer to get to our final, glorious IP address.

CNAMEs also have limits. For example, we can't use a CNAME record on the apex (`@`) domain. If you want to point the plain `donkeyrentals.com` to another domain with DNS, you're out of luck (with the exception of ALIAS records, see below). Also

CNAME records can never exist with other record types for the same hostname. Imagine our `www` pointing to `rabbitrentals.com` with a CNAME record, and *also* to our server's IP address with an A record. We have no way of knowing whether to use the CNAME record or A record, and each could have different outcomes.

For the most part, A records are the way to go when we want to connect a domain to a website or service. But sometimes CNAMEs are the better (or only) choice. If we want an honest-to-goodness alternate name for the same domain, CNAMEs are it. Or, for services like Heroku, we need to use a CNAME to point our domain to an app we created. Heroku hosts lots of websites on many servers where the IP addresses can change at any time, so there's no IP address where we can point an A record.

See the Common Scenario on using a CNAME on the apex domain to learn more.

CNAMEs vs Subdomains

There's a lot of confusion about the difference between a CNAME and a subdomain (at least there was for me when I started learning). They are two separate concepts, and it's important to understand both.

We might hear `www` talked about as a subdomain of `donkeyrentals.com` or as a CNAME record for the `donkeyrentals.com` domain. But which one is it?

Technically, `www` by itself is a hostname (referred to in the RFC 882 with a space as in "host name"). Hostnames are neither subdomains nor CNAME records. The subdomain is the full `www.donkeyrentals.com`, and `www` is the hostname at the beginning. (It's common to call `www` the subdomain even if that's technically incorrect.)

There's *also* the `www` entry in our domain's DNS records. It's not unusual for this to be a CNAME record, but it doesn't need to be. We can have `www` be an A record, NS record, or any other kind of record. Speaking of NS records...

NS

Nameserver (NS) records are the first point of contact to the outside world we as domain administrators are responsible for. In fact, the `dig` command we've been using is asking the nameservers to get its information. NS records are often

neglected when setting up or changing a domain. Most people just skip right to those attractive and charismatic A and CNAME records.

So what do NS records point to? Each one points to a server that holds all the records we create. Usually, we'll configure two or more nameservers when setting up a domain. That way, if one is offline, the other can pick up the slack. Here's how it might look:

- **Nameserver 1**: `ns1.example.com`
- **Nameserver 2**: `ns2.example.com`
- **Nameserver 3**: `ns3.example.com`

Nameservers are often in the form of `nsX.example.com` where `X` is a number starting at 1 and `example.com` is the website where you registered your domain. This isn't a standard, just a convention. If you had to guess what the nameservers are, start here.

In all the domain tech support I've done, wrongly set nameserver records are the most common source of confusion for newcomers. If they're set wrong, nothing will work because the nameserver finds all the other records in your domain. If you find that your domain points to some placeholder page, it's likely that your name servers are wrong.

Imagine you were looking someone up in a phonebook for the wrong town. You wouldn't be able to find that person. The phonebook here is the nameserver, and each person here is a record. Wrong nameserver, wrong records.

We can check our configuration with `dig`:

```
$ dig +short donkeyrentals.com NS

ns1.hover.com
ns2.hover.com
```

This isn't as useful as you might think. Since `dig` uses the nameservers to get information, asking for those nameservers is a bit cyclical. It's like calling someone up to ask them for their phone number. Looking up NS records with `dig` is still useful to find out if your nameservers are incorrect, but it's not useful for figuring out the right ones.

TXT

TXT (text) records are incredibly simple. They are a string of text connected to a hostname. But since they are so simple and their purpose is undefined, they are very much the "other" type of record. For example:

- **Hostname**: `message`
- **Record Type**: `TXT`
- **Value**: `Welcome to Donkey Rentals!`

```
$ dig message.donkeyrentals.com TXT +short

"Welcome to Donkey Rentals!"
```

So, what's the point? Why would we want these? Sometimes extra metadata is useful. We can use it to prove that we do, in fact, have control over this domain. Email services use TXT records to help prove that email coming from our email addresses is actually from us. Spammers can't fake our email address so easily then. This is also true for SSL/TLS certificates.

Another example: clever programmers have used TXT records to extend what DNS records can do. In the CNAME section above, we were talking about how we can't use CNAMEs on the apex domain. ALIAS records (see below) allow us to do this, but ALIAS records aren't standard. It's all a clever use of TXT records.

SRV

SRV (service) records exist to make it easier to connect to a specific service on a specific port. By "service" I mean something other than a website, such as Minecraft. It uses SRV records to help players to connect to a multiplayer online Minecraft game by typing `play.donkeyrentals.com` instead of `92.112.56.78:9000`. SRV records can also provide server prioritization and load balancing. We'll get into all this below.

Our pretend Minecraft server runs on a machine just like our website does. We can contact it at an IP address `92.112.56.78` and on a specific port `9000`. Normally, players would have to enter that IP address and port number to connect to our

server, but all those numbers are a pain to remember. Good thing DNS is, in fact, built to solve this problem.

Instead of an IP address, we'll have our players connect to the server at `play.donkeyrentals.com`. We'll need to set up an A or AAAA record for the `play` hostname. We'll set up the SRV record next, but first we need to have a few pieces of information.

When defining how users connect to a service, each developer (in this case, Minecraft) defines certain aspects of the service, such as the **name** and **protocol**. The name for Minecraft is, unsurprisingly, `minecraft`, and the protocol is `tcp`. These pieces of data get an underscore in front of them. Put them together and we get: `_minecraft._tcp`.

Next, we add the first part of the domain we came up with earlier: `play`. This should not have an underscore and comes right after the protocol: `_minecraft._tcp.play`. Are you ready to set up the DNS record yet? Nope. Too bad! We still have two more things to talk about: **priority** and **weight**.

Priority and weight

Briefly: priority chooses one server over another and weight distributes the load between multiple servers.

Less briefly: If a service has more than one SRV record for primary and backup servers, **priority** is a way to specify that we want one server used before trying another. For example, normally we want people to just use the primary server. But if it's offline, we can use the backup server instead. We'll start with the lowest number and work our way up until a server responds. Because of this "lowest first" mechanism, priority is often called distance or preference. Priority is a number between 0 and 65535, in case you're a rich person and have 65,000+ servers.

Weight is similar, but works differently. Just like priority, it's a number between 0 and 65535. Instead of a server getting used before another, the load will spread out according to the weight value. Weight only applies when there are two or more SRV records with the same priority.

Imagine we have two containers of different sizes for collecting water, a bucket and a cup. They're both perfectly fine containers that we can use at the same time (same priority) but `bucket` is twice the size of `cup`. When we fill them both to

capacity, we'll put two drops of water in `bucket` for every drop we put in `cup`. So the **weight** for `bucket` will be `2` and `cup` will be `1`. This is how weights in SRV records work, except instead of drops of water, it's connections to the server. (Water is not great for servers.)

Ok, *now* we can set up our SRV record:

- **Hostname**: `_minecraft._tcp.play`
- **Record Type**: `SRV`
- **Priority**: `1`
- **Weight**: `1`
- **Port**: `9000`
- **Value**: `play.donkeyrentals.com`

```
$ dig +short _minecraft._tcp.play.donkeyrentals.com SRV

1 1 9000 play.donkeyrentals.com.
```

Note the `play` in the **Hostname** relates to the `play` in the **Value**. Whatever users type to get to your server (e.g. `play.donkeyrentals.com`), the first part must match up to the last part of the **Hostname**.

In this example, I chose `1` for the **Priority** and **Weight** because, when we only have one server, it doesn't matter which one gets picked first or how often it's used.

This isn't the only way to set up an SRV record, as they can vary widely by service. We'll want to check the documentation from the provider (e.g., Minecraft) to make sure we've set ours up the right way.

MX

MX (Mail eXchange – the X looks cool that way) records are for email. In the same way A or CNAME records point to a website, MX records tell email where to go on the internet. If you break apart an email address like `eeyore@donkeyrentals.com`, you have two parts: `eeyore` and `donkeyrentals.com`. MX records are only concerned with the second part: `donkeyrentals.com`.

Your email provider (Gmail, Hotmail, Fastmail, etc.) will have a list of servers you need to make MX records for. First, let's try to set one up:

- **Hostname**: `@`
- **Record Type**: `MX`
- **Priority**: `10`
- **Value**: `aspmx.l.google.com`

This is part of the configuration for Gmail. We'll walk through it step by step.

Remember, the `@` in the **Hostname** is for the apex domain and has nothing to do with the `@` in an email address. It represents the domain without anything in front of it, i.e., `google.com` not `www.google.com`. We're saying we want to configure the server for emails that get sent to `<anything>@donkeyrentals.com`. We're not concerned about email sent to other subdomains, just this one.

The **record type** (`MX` in this case) should be pretty obvious. We're talking about `MX` records after all.

Priority is the same as in SRV records. A lower number means this server will be used before any other server is tried. There can be multiple mail servers for a particular domain. If one server is down or unavailable, it will try the next one in the list.

Finally, the **Value** is the domain name of a server. This configuration might look similar to a CNAME record, but the domain name we point to must be an A or AAAA record, i.e., we can't point an MX record to a CNAME or other type of record. It has to go to a domain that points directly to an IP address.

We can now verify the record:

```
$ dig +short donkeyrentals.com MX

10 aspmx.l.google.com
```

How email uses MX records

Let's take a second to talk about how email uses these records. When an email server needs to deliver to `donkeyrentals.com`, it looks up MX records for that domain. We just did the same thing for our domain. But let's pretend there are a

bunch of servers instead of just one. This is a much more common scenario:

```
$ dig +short donkeyrentals.com MX

5 gmail-smtp-in.l.google.com.
10 alt1.gmail-smtp-in.l.google.com.
20 alt2.gmail-smtp-in.l.google.com.
30 alt3.gmail-smtp-in.l.google.com.
40 alt4.gmail-smtp-in.l.google.com.
```

The mail server first tries the record with the lowest priority on the list, `gmail-smtp-in.l.google.com.`, and looks up its IP address. Remember that all MX records must point to an A or AAAA record, so they will always resolve to an IP address:

```
$ dig +short gmail-smtp-in.l.google.com A

173.194.205.27
```

Then it tries to send the email data. If it fails, it moves on to the item with the next server `alt1.gmail-smtp-in.l.google.com.`, which is next lowest in priority. The mail server tries server after server until either one succeeds or they all fail.

It might seem weird to use the server with the *lowest* priority first. For that reason, priority is often called **distance**. That way, we can think of it as starting with closer servers first then moving outward. Of course, this has nothing to do with the physical location of the servers. It's just a mental model to help remember how this number works.

ALIAS or ANAME

These records are a special case because, as of this writing, very few DNS providers implement them. When they are implemented, their names and implementations vary wildly. So why are we even talking about them? Because they let us do a very special thing: point the apex domain to a non-address record.

"But didn't you say that the apex domain has to point to an IP address?" Yes, I lied. Sort of. It's technically true; according to long and boring documents, the apex domain *must* not use a CNAME record. Despite this, some DNS hosts have started creating new features outside these rules.

DNSimple is using TXT records to point apex domains just like a CNAME does. They call this an ALIAS record. DNS Made Easy has a similar feature called ANAMEs. I want to stress again that these are *non-standard* uses of DNS technology only available through limited DNS providers. Each providers' implementation is entirely different. They make me nervous.

Even so, they can be useful. For example, I mentioned that Heroku instructs customers to use these records so an apex domain can point toward a Heroku app.

Since these aren't the same across the board, I can't promise this is how we'll have to set one up. But it will likely work something like this:

- **Hostname**: `@`
- **Record Type**: `ALIAS`
- **Target Host**: `someotherwebsite.com`

When we want to test our setup, if we're lucky, our DNS provider will let us ask it about ALIAS or ANAME records directly:

```
$ dig +short donkeyrentals.com ALIAS

104.131.191.2
```

But again, these are *non-standard* record types, so we may have to resort to alternate methods:

```
$ dig +short donkeyrentals.com TXT

"ALIAS for someotherwebsite.com"
```

Notice we looked for a TXT record here. That's how DNSimple does it, but it might work some other way entirely! That's the brave new world of non-standard records. Exciting, no? (ALIAS records. Coming soon to a theater near you.)

What If It All Goes Wrong?

We covered a lot of ground in this chapter. If everything went well, we have all sorts of records pointing at many servers. But what if we screwed up? (Don't worry, we will.) How can we fix it? Even in a best-case scenario, how can we get precise information about our domains? Tune in next chapter to find out what happens in *Daring Domains Debugged!*

Tools of the Trade

How DNS Works

There are many steps between typing a URL into our browser and seeing the website. It's all very complicated, but there's a method to the madness. To debug our domains, we'll need a quick primer in the deep, dark ways of the domain name system.

Root Servers

All domains start somewhere, and that somewhere is the root servers. There are 13 root servers labeled `a` through `m`. To see them, simply type `dig +short`:

```
dig +short

a.root-servers.net.
b.root-servers.net.
c.root-servers.net.
d.root-servers.net.
e.root-servers.net.
f.root-servers.net.
g.root-servers.net.
h.root-servers.net.
i.root-servers.net.
j.root-servers.net.
```

```
k.root-servers.net.
l.root-servers.net.
m.root-servers.net.
```

Told you.

These are nameservers. The order they are in doesn't matter. We can use any of them. Each one stores the information needed to contact all top-level domains like `com`, `net`, `horse`, etc. The `com` nameservers in turn store information about Hover, the DNS provider I use for donkey rentals. Finally, hover has information about donkeyrentals.com.

That's how DNS works: each server looks in its database to find out how to get to the next level below it. To get to any domain, we have to go through all these steps. Well, *we* don't have to go through all those steps. Software called a resolver does.

The Resolver

The piece of software that actually makes these requests is called a DNS resolver. Its job is to turn, or resolve, `donkeyrentals.com` into `104.131.191.2` by taking all the steps above. Tools like dig let us see what the resolver sees so we can confirm our DNS settings.

Dig lets us act like a resolver. We can see the whole chain of server interactions laid bare, which then lets us see where things are misconfigured. For example, if we're expecting the nameservers for `donkeyrentals.com` to be `ns1.equestrian-domains.com` and they are, in fact, `ns1.bovinian-domains.com`, we have a problem. If we expect `www.donkeyrentals.com` to be a CNAME that points to `donkeyrentals.com`, but it instead points to `buzzfeed.com`, we have a different problem.

Caching

The other thing to know about DNS is that each step in the resolution process gets cached. When the resolver asks `com` for information about `donkeyrentals.com`, it gets stored for some amount of time, which can be seconds to days, depending on the server.

The reasoning here is simple: if the whole chain was traced every time anyone in the entire world wanted to go to a website, it would put an enormous load on the root servers. With only 13 root servers, that would be billions of requests a day. In reality, each letter points to multiple servers. But even if there were 500+, that's still too much traffic for a server to handle.

That should be enough to get us started with my very favorite debugging tool: dig.

dig

So far, we've been using the dig command throughout this book to check our DNS handiwork. It stands for Domain Information Groper (ew), and its sole job is to get information about DNS records. This thing is our Swiss Army knife for DNS debugging.

If you don't have dig installed on your machine, you can install it. If that seems to complicated, internet nerds have recreated dig's functionality on the web. Search for "dig web interface" and you should find a few tools.

Dig Basics

The first step with dig is to look up A records:

```
$ dig donkeyrentals.com

; <<>> DiG 9.8.3-P1 <<>> donkeyrentals.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 26538
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;donkeyrentals.com.     IN  A

;; ANSWER SECTION:
donkeyrentals.com.  842 IN  A   104.131.191.2
```

```
;; Query time: 7 msec
;; SERVER: 192.168.128.1#53(192.168.128.1)
;; WHEN: Thu Jun  2 10:16:46 2016
;; MSG SIZE  rcvd: 51
```

The command breaks down like this:

- `dig` – The command.
- `donkeyrentals.com` – The domain we'd like information for.

The response breaks down like this:

- Header – We can safely ignore this for now. For the curious, this section shows us information about the response, such as what type of response it is (`opcode: QUERY`), what flags are present (`flags: qr rd ra;`), and what types of responses we got (`QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0`).
- Question section – Also unimportant. It's only confirming what we asked for: A records for `donkeyrentals.com`.
- Answer section – This is the part we care about because it contains the information we asked for.
- Statistics – This is meta information about our request, such as how long it took, how large it was, and what server was queried. Not important for now, but this will come up later.

(Now, I know you're thinking, "Wow, this is captivating reading! Where can I learn even more?!" RFC 1035 from the Internet Engineering Task Force is a great place to start if you really want to dive into the weeds here.)

Other Record Types

By default, dig looks up A records for our domain (just the apex domain, not any subdomains) and, as we can see, it returned `104.131.191.2`, the IP address of our apex domain's A record. We can look up other record types as well:

```
$ dig message.donkeyrentals.com TXT

; <<>> DiG 9.8.3-P1 <<>> message.donkeyrentals.com TXT
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 50384
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;message.donkeyrentals.com. IN  TXT

;; ANSWER SECTION:
message.donkeyrentals.com. 900  IN  TXT "Welcome to Donkey Rentals!"

;; Query time: 22 msec
;; SERVER: 192.168.128.1#53(192.168.128.1)
;; WHEN: Thu Jun  2 10:17:36 2016
;; MSG SIZE  rcvd: 82
```

Hidden in the response here, we can see the TXT record ("Welcome to Donkey Rentals!") we added in the previous chapter. Any of the other record types we talked about there are available too.

Query Options

Up until this chapter, I've been using dig with the `+short` query option to just get the bare minimum information we needed. Dig's query options all have a `+` at the start:

```
$ dig +short donkeyrentals.com

104.131.191.2
```

What order these options come in does matter. Always put the query option before the domain we're querying for. I've seen some wacky output when the query option comes after the domain.

The dig manual lists many query options besides `+short` we can look through at our leisure, but for now I'll point out a few of the more useful ones:

- `+trace`

At the beginning of this chapter, we talked about how DNS queries travel through a chain of servers before finally getting to our domain. This will illustrate that whole process. Try it!

- `+noquestion`, `+noanswer`, `+noadditional`, and `+noauthority`

Query options can also have `no` at the start to remove some functionality. These query options are all ways to limit the response dig gives back to us. If we never care about the `QUESTION` section, for example, we can simply turn it off and make dig's response much less wordy.

Common Use Cases

Just the answer section

Dig has a query option called `+noall` that, as we might expect, turns off all output. It's not very useful by itself, but combined with a different query option such as `+answer`, gives us only the sections we want:

```
$ dig +noall +answer donkeyrentals.com A

donkeyrentals.com.  900 IN  A   104.131.191.2
```

Records without the cache

As we know, there's lots of caching involved in DNS. Since all of our DNS records exist on a nameserver, we can ask that nameserver directly and bypass any caching information:

```
$ dig @ns1.hover.com donkeyrentals.com A

; <<>> DiG 9.8.3-P1 <<>> @ns1.hover.com donkeyrentals.com A
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 44948
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
;; WARNING: recursion requested but not available

;; QUESTION SECTION:
;donkeyrentals.com.     IN  A

;; ANSWER SECTION:
donkeyrentals.com.  900 IN  A   104.131.191.2

;; Query time: 41 msec
;; SERVER: 216.40.47.26#53(216.40.47.26)
;; WHEN: Thu Jun  2 10:19:58 2016
;; MSG SIZE  rcvd: 51
```

In this case, we ask the name server `ns1.hover.com` what the A records are for `donkeyrentals.com`. You might think, "Isn't this what we've been doing all along?" Yes and no. The first time we ask for `donkeyrentals.com`, it gets the uncached answer from `ns1.hover.com`, but then our DNS server (or network router, or internet service provider, or myriad other places) stores it in their cache for a certain number of seconds, represented by the `900` (15 minutes) in the answer line. That number, by the way, is called *Time to live* or *TTL*.

Theoretically, after those 15 minutes pass, we'll get the uncached answer again, or at least the cache will have updated to display more up-to-date information if anything has changed.

To force our domain resolver to get the latest information, we can say, "No, do not ask our caches, ask the server directly for information." We can tell that we got an authoritative answer by looking at the flags in the header: `flags: qr aa rd`. That `aa` stands for **A**uthoritative **A**nswer. Also, in the stats at the bottom, we can see that we queried the server at `216.40.47.26`, which is actually what the A record for `ns1.hover.com` points to. If we don't tell it which server to query, it will use the DNS

server we have configured for our computer.

Multiple Queries at Once

If we have a bunch of queries to make, dig has a batch mode. First, we'll need a file with a list of queries. This list will look a lot like our dig commands so far, but there will be no `dig` at the start, and each query will have its own line:

dig.txt

```
@ns1.hover.com donkeyrentals.com
www.donkeyrentals.com CNAME
+short donkeyrentals.com NS
```

The, we use dig with the `-f` flag:

```
$ dig -f dig.txt

... not gonna paste all the responses here ...
```

This will make all the queries in order. Feel free to use query options here, too, like `+short` to make all queries have short responses:

```
$ dig +short -f dig.txt

... no really, pixels are surprisingly expensive ...
```

Looking Up a Domain Name by It's IP Address

So far, we've been looking up domain names to get their IP addresses, but can we do this in the other direction? We can indeed. This is what the `-x` flag is for:

```
$ dig +short -x 66.220.156.2

edge-star-shv-07-ash4.facebook.com.
```

This returns a PTR record. We didn't cover these in the last chapter because we don't ever get to actually configure them. Briefly, PTR stands for pointer and they *point* back to the original domain. In this case, we can see that it's a server at `facebook.com`.

Now, keep in mind this doesn't point back to a website, but to a server. If `somewebsite.com` is hosted at `somehost.com` with many other websites, a reverse lookup is more likely to show `someserver.somehost.com` rather than `somewebsite.com`. Facebook is so large they host their website on many servers, so they're easier to track down. Since it's not always accurate, I use these reverse lookups more as a hint than an answer.

Another anecdote is that the "reverse DNS lookup" database is hosted at `in-addr.arpa` domain. All lookups happen by reversing the sections of the ip address and prepending the result to `in-addr.arpa`. So, if you're trying to look up `12.34.56.78`, you can also use `dig 78.56.34.12.in-addr.arpa PTR`.

Curiosities

What does the `IN` mean?

In dig responses, we often see `IN` in the response:

```
;; QUESTION SECTION:
;donkeyrentals.com.     IN  A
```

This doesn't mean "in" like "A records all up **in** ya donkeyrentals" but is short for "Internet". Turns out all records have a class. Other classes are like an entirely separate internet and have nothing to do with this book. For our uses, we are always using the IN (Internet) class. That's also the class that dig defaults to. It can also be CH (Chaos) or HS (Hesiod), but we won't be talking about these no matter how rad they sound.

Why do records get returned in a different order?

When we ran dig to see all the root servers, they may have come back in a seemingly random order. This is called round-robin DNS. When we get back a list of IP

addresses from a DNS query, a domain resolver will generally start with the first address in the list. If the first server doesn't respond, it will pick the next one, and so on.

To make sure one server doesn't take all the heat, DNS providers can change the order in which the records are returned. This helps distribute requests across multiple servers. Not all DNS providers do this, and it's not always done in the same way, but keep an eye out for it.

Why does the TTL value change drastically?

While writing this book, this was a common scenario I ran into:

```
$ dig somedomain.com A

somedomain.com. 127 IN  A   12.34.56.78

$ dig somedomain.com A

somedomain.com. 125 IN  A   12.34.56.78

$ dig somedomain.com A

somedomain.com. 541 IN  A   12.34.56.78
```

Dig responses shortened to save pixels. Consider donating to the Save-A-Pixel foundation for America.

What's going on here? We made a request to get the A records at `somedomain.com`. We see the TTL for the first request is `127`. Then, a couple of seconds later, we make the same request and see that the TTL has decreased by 2 seconds to `125`. Seems about right. Finally, after another couple of seconds, we make the request a third time and get `541`. What?!

This is a perfect example of round-robin DNS. Just as I described above, DNS providers change the order of DNS records. So, in this case, let's say there were looking at two servers, A and B. We saw the TTL value for server A, and then again

for server A after it had decreased slightly. Then we saw server B, which has a totally different TTL value.

It's pretty normal to see this, so don't let it phase you, but it is certainly jarring and confusing the first time.

nslookup

I like dig. It's a good tool for debugging, probably the best. But one problem with dig is that it's not installed on the Windows operating system by default. That's a bummer if we need to do some quick DNS troubleshooting. Sure, you could use one of dig's many online interfaces, but it doesn't have that new command line smell.

There's another tool called nslookup that, for the most part, does everything dig does, but there are a couple caveats.

Nslookup is a part of BIND, a suite of software tools for running DNS servers. BIND includes a bunch of tools, including nslookup and dig. ISC, the company that maintains BIND, states in the BIND manual, page 9:

> Due to its arcane user interface and frequently inconsistent behavior, we do not recommend the use of nslookup. Use dig instead.

Talk about an authoritative answer! (Wink to camera.)

That's not all, though. nslookup performs other unnecessary queries in the background. Not only are these useless, but they can actually screw up the response nslookup returns. This won't matter for most requests, but if those initial, unnecessary queries fail, it can stop our query in its tracks.

These flaws are unfortunate because alternate tools can be very useful. But I'd recommend you avoid nslookup altogether. You can read more about the flaws in nslookup here and here.

Basic Usage

Anyway, if you skipped the last few paragraphs let's start with the most basic example, looking up A records for `donkeyrentals.com`:

```
$ nslookup
> donkeyrentals.com
Server:     192.168.128.1
Address:    192.168.128.1#53

Non-authoritative answer:
Name:   donkeyrentals.com
Address: 104.131.191.2
```

The biggest difference with nslookup is the way you use it. Type `nslookup` and hit enter. This brings up what's called interactive mode. You can type commands here without having to type nslookup first. The command prompt here is indicated with `>` instead of `$`. Type `exit` or hit `ctrl-c` to leave this mode and go back to the terminal.

There is a way to use nslookup without single, one-off commands, but by far the most common way I've seen it used is in this interactive mode.

We can type domain names and get their A records, just like dig. That's the first step. Here are some examples of other queries:

Getting a Little More Info

The `debug` setting can give you a more dig-like response, including the question and TTL if you want them:

```
> set debug
> donkeyrentals.com
Server:     192.168.128.1
Address:    192.168.128.1#53

------------
    QUESTIONS:
    donkeyrentals.com, type = A, class = IN
    ANSWERS:
    ->  donkeyrentals.com
    internet address = 104.131.191.2
    ttl = 576
```

```
    AUTHORITY RECORDS:
    ADDITIONAL RECORDS:
------------
Non-authoritative answer:
Name:   donkeyrentals.com
Address: 104.131.191.2
```

To turn this off, type `set nodebug` at the prompt.

Look Up Other Record Types, Such As NS

```
> set type=ns
> donkeyrentals.com
Server:     192.168.128.1
Address:    192.168.128.1#53

Non-authoritative answer:
donkeyrentals.com   nameserver = ns1.hover.com.
donkeyrentals.com   nameserver = ns2.hover.com.

Authoritative answers can be found from:
ns2.hover.com   internet address = 64.98.148.13
ns1.hover.com   internet address = 216.40.47.26
```

We `set` the `type` to `ns`, `cname`, `mx`, or any of the other types. We can use `set type=` again to set it to a different type.

Get an Authoritative Answer

We can set the server similarly to how we specify a server with dig:

```
> server ns1.hover.com
Default server: ns1.hover.com
Address: 216.40.47.26#53
```

```
> donkeyrentals.com
Server:     ns1.hover.com
Address:    216.40.47.26#53

donkeyrentals.com   nameserver = ns1.hover.com.
donkeyrentals.com   nameserver = ns2.hover.com.
```

This attempts to query a nameserver directly instead of using cached information.

Looking Up a Domain Name by Its IP Address

This one is super simple. Just enter the IP address:

```
> 66.220.156.2
Server:     192.168.128.1
Address:    192.168.128.1#53

Non-authoritative answer:
2.156.220.66.in-addr.arpa   name = edge-star-shv-07-ash4.facebook.com.
```

That's nslookup in a nutshell. It can be useful, but I would still avoid it. It's not worth hours of frustration because it interpreted a response wrong or failed when it shouldn't have. Stick to dig if you can.

WHOIS

Let's clear up one thing real fast: WHOIS in all caps is a protocol, while `whois` in all lowercase is a command line tool. They are related but not the same thing. The `whois` tool uses the WHOIS protocol to retrieve information about a domain. We'll be interacting with `whois` directly and WHOIS indirectly.

The information we get using `whois` is different for every domain. The most common and useful pieces include contact information for the person or company that registered the domain and/or the expiration date of that domain. The catch is that

this information has no standard to it. It can be any information, in any format. As we'll see, this can make working with WHOIS incredibly annoying.

Let's take a look at a response. It's a long one:

```
Whois Server Version 2.0

Domain names in the .com and .net domains can now be registered
with many different competing registrars. Go to http://www.internic.net
for detailed information.

   Domain Name: DONKEYRENTALS.COM
   Registrar: TUCOWS DOMAINS INC.
   Sponsoring Registrar IANA ID: 69
   Whois Server: whois.tucows.com
   Referral URL: http://www.tucowsdomains.com
   Name Server: NS1.HOVER.COM
   Name Server: NS2.HOVER.COM
   Status: clientTransferProhibited https://www.icann.org/epp#clientTransferProhibited
   Status: clientUpdateProhibited https://www.icann.org/epp#clientUpdateProhibited
   Updated Date: 20-jan-2016
   Creation Date: 23-sep-2015
   Expiration Date: 23-sep-2016

>>> Last update of whois database: Wed, 20 Jan 2016 18:09:25 GMT <<<

For more information on Whois status codes, please visit
https://www.icann.org/resources/pages/epp-status-codes-2014-06-16-en.

NOTICE: The expiration date displayed in this record is the date the
registrar's sponsorship of the domain name registration in the registry is
currently set to expire. This date does not necessarily reflect the expiration
date of the domain name registrant's agreement with the sponsoring
registrar.  Users may consult the sponsoring registrar's Whois database to
view the registrar's reported date of expiration for this registration.

TERMS OF USE: You are not authorized to access or query our Whois
database through the use of electronic processes that are high-volume and
```

automated except as reasonably necessary to register domain names or
modify existing registrations; the Data in VeriSign Global Registry
Services' ("VeriSign") Whois database is provided by VeriSign for
information purposes only, and to assist persons in obtaining information
about or related to a domain name registration record. VeriSign does not
guarantee its accuracy. By submitting a Whois query, you agree to abide
by the following terms of use: You agree that you may use this Data only
for lawful purposes and that under no circumstances will you use this Data
to: (1) allow, enable, or otherwise support the transmission of mass
unsolicited, commercial advertising or solicitations via e-mail, telephone,
or facsimile; or (2) enable high volume, automated, electronic processes
that apply to VeriSign (or its computer systems). The compilation,
repackaging, dissemination or other use of this Data is expressly
prohibited without the prior written consent of VeriSign. You agree not to
use electronic processes that are automated and high-volume to access or
query the Whois database except as reasonably necessary to register
domain names or modify existing registrations. VeriSign reserves the right
to restrict your access to the Whois database in its sole discretion to ensure
operational stability. VeriSign may restrict or terminate your access to the
Whois database for failure to abide by these terms of use. VeriSign
reserves the right to modify these terms at any time.

The Registry database contains ONLY .COM, .NET, .EDU domains and
Registrars.
Domain Name: DONKEYRENTALS.COM
Registry Domain ID: 1962822874_DOMAIN_COM-VRSN
Registrar WHOIS Server: whois.tucows.com
Registrar URL: http://tucowsdomains.com
Updated Date: 2015-09-23T18:00:04Z
Creation Date: 2015-09-23T18:00:04Z
Registrar Registration Expiration Date: 2016-09-23T18:00:04Z
Registrar: TUCOWS, INC.
Registrar IANA ID: 69
Registrar Abuse Contact Email: domainabuse@tucows.com
Registrar Abuse Contact Phone: +1.4165350123
Reseller: Hover
Domain Status: clientTransferProhibited
Domain Status: clientUpdateProhibited

```
Registry Registrant ID:
Registrant Name: Contact Privacy Inc. Customer 0141386251
Registrant Organization: Contact Privacy Inc. Customer 0141386251
Registrant Street: 96 Mowat Ave
Registrant City: Toronto
Registrant State/Province: ON
Registrant Postal Code: M6K 3M1
Registrant Country: CA
Registrant Phone: +1.4165385457
Registrant Phone Ext:
Registrant Fax:
Registrant Fax Ext:
Registrant Email: donkeyrentals.com@contactprivacy.com
Registry Admin ID:
Admin Name: Contact Privacy Inc. Customer 0141386251
Admin Organization: Contact Privacy Inc. Customer 0141386251
Admin Street: 96 Mowat Ave
Admin City: Toronto
Admin State/Province: ON
Admin Postal Code: M6K 3M1
Admin Country: CA
Admin Phone: +1.4165385457
Admin Phone Ext:
Admin Fax:
Admin Fax Ext:
Admin Email: donkeyrentals.com@contactprivacy.com
Registry Tech ID:
Tech Name: Contact Privacy Inc. Customer 0141386251
Tech Organization: Contact Privacy Inc. Customer 0141386251
Tech Street: 96 Mowat Ave
Tech City: Toronto
Tech State/Province: ON
Tech Postal Code: M6K 3M1
Tech Country: CA
Tech Phone: +1.4165385457
Tech Phone Ext:
Tech Fax:
Tech Fax Ext:
```

```
Tech Email: donkeyrentals.com@contactprivacy.com
Name Server: NS1.HOVER.COM
Name Server: NS2.HOVER.COM
DNSSEC: unsigned
URL of the ICANN WHOIS Data Problem Reporting System: http://wdprs.internic.net/
>>> Last update of WHOIS database: 2015-09-23T18:00:04Z <<<

Registration Service Provider:
    Hover, help@hover.com
    +1.8667316556
    http://help.hover.com


This domain's privacy is protected by contactprivacy.com. To reach the domain contacts,
please go to http://www.contactprivacy.com and follow the instructions.

The Data in the Tucows Registrar WHOIS database is provided to you by Tucows
for information purposes only, and may be used to assist you in obtaining
information about or related to a domain name's registration record.

Tucows makes this information available "as is," and does not guarantee its
accuracy.

By submitting a WHOIS query, you agree that you will use this data only for
lawful purposes and that, under no circumstances will you use this data to:
a) allow, enable, or otherwise support the transmission by e-mail,
telephone, or facsimile of mass, unsolicited, commercial advertising or
solicitations to entities other than the data recipient's own existing
customers; or (b) enable high volume, automated, electronic processes that
send queries or data to the systems of any Registry Operator or
ICANN-Accredited registrar, except as reasonably necessary to register
domain names or modify existing registrations.

The compilation, repackaging, dissemination or other use of this Data is
expressly prohibited without the prior written consent of Tucows.

Tucows reserves the right to terminate your access to the Tucows WHOIS
database in its sole discretion, including without limitation, for excessive
```

```
querying of the WHOIS database or for failure to otherwise abide by this
policy.

Tucows reserves the right to modify these terms at any time.

By submitting this query, you agree to abide by these terms.

NOTE: THE WHOIS DATABASE IS A CONTACT DATABASE ONLY.  LACK OF A DOMAIN
RECORD DOES NOT SIGNIFY DOMAIN AVAILABILITY.
```

Hello. Welcome to, like, three pages later.

There's a lot to this response, as we can see. Most of it is legal disclaimer because WHOIS information has been the source of a lot of legal trouble over the years. The act of publicly displaying a customer's personal information has (surprise!) caused a lot of problems.

There's also some meta-information about the servers that were contacted to get this information, and also some pseudo-advertisements. For the most part, you can ignore these.

Toward the middle is the information we're probably looking for: to whom the domain is registered and when it expires. This can give us leads for who to contact if there's a problem or when the domain might go up for sale.

Different TLDs

Above is just one example of what a `com` domain will return but, as I said, each TLD is different. Here's a different response from a `gov` domain:

```
$ whois nasa.gov

% DOTGOV WHOIS Server ready
   Domain Name: NASA.GOV
   Status: ACTIVE

>>> Last update of whois database: 2016-01-20T19:12:13Z <<<
Please be advised that this whois server only contains information pertaining
```

```
to the .GOV domain. For information for other domains please use the whois
server at RS.INTERNIC.NET.
```

Much shorter! This is because there's no guide or specification that dictates what WHOIS information needs to include or how it should be formatted. What a great "standard" this is!

Different WHOIS Servers

Another aspect to consider: since all this information is stored on different servers (all TLDs are run by different companies), occasionally we'll need to specify which server to use. For example, if we try to look up a `dentist` domain, we get:

```
$ whois donkey.dentist

whois: dentist.whois-servers.net: nodename nor servname provided, or not known
```

It can't find the right server. Luckily the root zone database list all TLDs and their WHOIS server info. Clicking on the dentist domain gives us a bunch of information about the TLD and, at the bottom, a WHOIS server: `whois.rightside.co`. Now we can make our WHOIS query using the `-h` option:

```
$ whois -h whois.rightside.co donkey.dentist

Domain not found.

...
```

Be right back. I need to go register a domain real quick.

Multiple Domains

This is a tricky one, but sometimes `whois` isn't smart enough to figure out what you mean. For example, if you try to get WHOIS information for a popular domain:

```
$ whois google.com

Aborting search 50 records found .....
GOOGLE.COM.AFRICANBATS.ORG
GOOGLE.COM.ANGRYPIRATES.COM
GOOGLE.COM.AR
GOOGLE.COM.AU
GOOGLE.COM.BAISAD.COM
GOOGLE.COM.BEYONDWHOIS.COM
...
```

What is all this? Did we mistakenly type only a subdomain from `africanbats.org`? Unlikely. It seems like WHOIS just isn't that smart at figuring out what we mean. So how do we get the information for just `google.com`?

Maybe you tried this out yourself and saw this piece of text in the gigantic domain list:

> To single out one record, look it up with "xxx", where xxx is one of the records displayed above. If the records are the same, look them up with "=xxx" to receive a full display for each record.

But if we try that advice, one of two things happen. If we try surround it in quotes, we get the same result. If we use the `=` "trick" it shows us WHOIS information for *all* of the domains on that list:

```
$ whois "=google.com"

Aborting search 50 records found .....
   Server Name: GOOGLE.COM.AFRICANBATS.ORG
   Registrar: TUCOWS DOMAINS INC.
   Whois Server: whois.tucows.com
   Referral URL: http://www.tucowsdomains.com


   Server Name: GOOGLE.COM.ANGRYPIRATES.COM
   IP Address: 8.8.8.8
```

```
   Registrar: NAME.COM, INC.
   Whois Server: whois.name.com
   Referral URL: http://www.name.com

...
```

This does get us the information we want, but come on. Really, WHOIS!? We don't want to see any of that. Can't we just get the single domain we asked for?

Turns out that if you prepend `domain` onto the front of the domain you want and put it all in quotes, this works:

```
$ whois "domain google.com"

Whois Server Version 2.0

Domain names in the .com and .net domains can now be registered
with many different competing registrars. Go to http://www.internic.net
for detailed information.

   Domain Name: GOOGLE.COM
   Registrar: MARKMONITOR INC.
   Sponsoring Registrar IANA ID: 292
   Whois Server: whois.markmonitor.com
   Referral URL: http://www.markmonitor.com
   Name Server: NS1.GOOGLE.COM
   Name Server: NS2.GOOGLE.COM

...
```

But fair warning here: this **doesn't work with all domains**. (There's no standard for WHOIS, remember? It can be anything! What a great syst–. Uhg, I can't finish that sentence with a straight face.)

There are alternatives, such as jwhois, that a lot of people really like. There are also web-based tools that help get around these issues. I recommend using one of those instead.

Honestly, the more I look into WHOIS, the less I am a fan of it. It can be useful for checking to see if a domain is registered, but overall, it's a mess. The Internet Engineering Task Force (which wins the *Best Company Name In This Book* award) and ICANN both recognize that WHOIS is flawed and are trying to replace it.

Hopefully, we'll see this happen, but for now WHOIS is all we have.

host

`host` is kind of like a simplified version of `dig`. In fact, it's developed by the same company that makes `dig` and `nslookup`. This will be a quick section since there's not much to the `host` tool, but it's a nice thing to have if you are looking for friendlier responses. Here's what I mean:

```
$ host donkeyrentals.com

donkeyrentals.com has address 104.131.191.2
```

Hard to be more friendly than that. That's the A record for `donkeyrentals.com` in simple English words. It can even interpret more complex responses:

```
$ host facebook.com

facebook.com has address 173.252.120.68
facebook.com has IPv6 address 2a03:2880:2130:cf24:face:b00c::25de
facebook.com mail is handled by 10 msgin.vvv.facebook.com.
```

Types

Of course, if we need to look up something other than A records, we can specify type with the `-t` option:

```
$ host -t CNAME www.donkeyrentals.com

www.donkeyrentals.com is an alias for donkeyrentals.com.
```

```
$ host -t NS donkeyrentals.com

donkeyrentals.com name server ns1.hover.com.
donkeyrentals.com name server ns2.hover.com.
```

Reverse Lookups

`host` can also do reverse IP lookups just like dig:

```
$ host -i 173.252.120.68

68.120.252.173.in-addr.arpa domain name pointer edge-star-mini-shv-12-frc3.facebook.com.
```

A Little More Info

Sometimes we do want a little more information than just a sentence. The `-v` option is helpful here:

```
$ host -v donkeyrentals.com

Trying "donkeyrentals.com"
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 49015
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;donkeyrentals.com.     IN  A

;; ANSWER SECTION:
donkeyrentals.com.  900 IN  A   104.131.191.2

Received 51 bytes from 192.168.128.1#53 in 37 ms
Trying "donkeyrentals.com"
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 65416
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
```

```
;donkeyrentals.com.     IN  AAAA

;; ANSWER SECTION:
donkeyrentals.com.  900 IN  AAAA    2620:0:861:ed1a::1

Received 63 bytes from 192.168.128.1#53 in 35 ms
Trying "donkeyrentals.com"
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 44277
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 2

;; QUESTION SECTION:
;donkeyrentals.com.     IN  MX

;; ANSWER SECTION:
donkeyrentals.com.  900 IN  MX  10 aspmx.l.google.com.

;; ADDITIONAL SECTION:
aspmx.l.google.com. 35  IN  A   209.85.201.27
aspmx.l.google.com. 267 IN  AAAA    2607:f8b0:400d:c04::1a

Received 110 bytes from 192.168.128.1#53 in 38 ms
```

Wait a second! This looks very familiar. In fact, it looks like they're just using `dig`! That makes some sense because, again, the same company developed both. In this case, we could probably just stick with `dig`, but it's good to know we have this option if we want it.

Why I Chose `dig` Instead

So, I like `host` enough to include a section about it. It feels nice to use sometimes when you want a quick answer, but I believe `dig` gives us more authentic information. It shows us in the response itself that something is an A record, or a CNAME, or NS, etc. This book is about learning those concepts, so it made the most sense to use `dig`. It's also the tool I see most used online in guides, tutorials, and Q&A sites. I hope familiarity with `dig` will help you continue to learn on your own after using this book.

However, `host` does pop up from time to time, and it can be a really nice alternative for something quick, but only when you first understand what it's telling you. `donkeyrentals.com has address 104.131.191.2` may look nice, but it doesn't give us a lot of information to work with.

ping & ping6

`ping` (and its sibling for IPv6 addresses, `ping6`) is a very simple tool, so this will be a short section. Its most common use case can be summed up as: "Are you there?" Although simple, this utility is useful in automated environments. Computers check to make sure other computers are awake, active, and receiving connections.

Its command line interface looks like this:

```
$ ping donkeyrentals.com

PING donkeyrentals.com (104.131.191.2): 56 data bytes
64 bytes from 104.131.191.2: icmp_seq=0 ttl=55 time=15.031 ms
64 bytes from 104.131.191.2: icmp_seq=1 ttl=55 time=21.293 ms
64 bytes from 104.131.191.2: icmp_seq=2 ttl=55 time=12.122 ms
64 bytes from 104.131.191.2: icmp_seq=3 ttl=55 time=12.928 ms
64 bytes from 104.131.191.2: icmp_seq=4 ttl=55 time=12.733 ms
^C
--- donkeyrentals.com ping statistics ---
5 packets transmitted, 5 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 12.122/14.821/21.293/3.381 ms
```

For IPv6 addresses, `ping6` is the tool to use:

```
$ ping6 2620:0:861:ed1a::1

...
```

It will run forever unless instructed to stop by typing the `CTRL-C` interrupt character. To limit the number of times it runs, we use the `-c` flag to set a "count":

```
$ ping -c 5 donkeyrentals.com

...
```

We can mostly ignore the details of the response as they're not particularly important to us. If we run `ping` for an extended period of time, the statistics at the bottom have useful information. Included are percentage of packets lost and combined times that each request took.

But mostly what we're looking for is either a successful response:

```
64 bytes from 104.131.191.2: icmp_seq=0 ttl=55 time=15.031 ms
```

or a failure:

```
Request timeout for icmp_seq 0
```

We can ignore the successes, but failures require attention. Specifically, the site is **down** and we need to work on bringing it back up. `ping` is great for these kinds of monitoring tasks.

Common Scenarios

In this chapter, I hope to cover some of the more common problems that I see people run into when trying to set up DNS and domains. There are a million different ways to configure websites and a million other different services that have their own requirements. It would be impossible to write them all up in detail, but I hope to give you the tools to find what you're looking for and troubleshooting tips for when it goes awry.

Feel free to skip to the sections you're interested in. If everything seems to work just fine, then skip the chapter altogether! Be warned though, you will miss a handful of mediocre jokes.

Creating a Subdomain

Hopefully, creating a subdomain is simple. If we have some kind of managed web hosting, there is often a control panel just for this purpose. The basic idea here is to create an A record that points to a server, exactly like our main website.

Subdomain for a Service

If this is for a service like an FTP server, we can point the `ftp` hostname to our FTP server using an A record. Anyone connecting over the file transfer protocol can use `ftp.donkeyrentals.com`. If we host our FTP server elsewhere, or we change where it's located, we can update that A record and the same domain will work.

Subdomain for the Web

The process is not too different from a service. If our hosting provider does not provide us with an easy way to make subdomains, we'll have a little more configuration to do.

The DNS setup is the same: we create an A record pointing to the IP address of a server. It's common, but not at all mandatory, to use the same server as our main website.

Once the request makes it to the server, our web server software–for this, we'll look at nginx and Apache–will take over and figure out what to do.

Let's say we're creating `store.donkeyrentals.com`. If we're using nginx, our server will need additional configuration to route requests to the store. Edit the nginx configuration for our website. Mine is located at `/etc/nginx/conf.d/default.conf`.

Each domain is divided up into a `server` directive. The directive for the store will look like this:

```
server {
  server_name store.donkeyrentals.com;
  root /var/www/donkey_store;
}
```

This short and sweet configuration can go above or below existing configurations. The `server_name` looks for requests coming in for that domain, and `root` specifies the directory to serve files from.

Apache is similar. Find our Apache configuration files (by default at `/etc/httpd/conf/httpd.conf`) and add a new `<VirtualHost>` directive:

```
<VirtualHost>
  ServerName store.donkeyrentals.com
  DocumentRoot /var/www/donkey_store
</VirtualHost>
```

We'll see striking similarities to the nginx configuration. `ServerName` will look for requests for `store.donkeyrentals.com`, and `DocumentRoot` will route those requests to files inside `/var/www/donkey_store`.

After either configuration, we'll need to restart nginx or Apache, respectively. Once that's done, our DNS can now work along side our web server. The A record points to an IP address, and the web server software will direct it to the correct files.

Transferring a Domain

Some domain registrars out there are... shall we say, not the best. Maybe we currently have a domain at one of these places, and we'd like to switch. Or we've purchased a domain from someone who uses a different domain registrar. Whatever the case, it's much easier to manage all our domains in one place.

To do that, we'll need to transfer our domain from one registrar to another. While every service is different, there are some standard steps that go along with every domain transfer. For the sake of this guide, our domain is currently residing at OldCrusty Domains. We'll be transferring from OldCrusty to our new registrar: ProbablyBetter Domains.

I'd recommend reading through this whole section before starting any transfer process. It's one of those topics that requires time and specific steps done in a specific order. The more you know before you start, the better.

Unlocking the Domain

If our domain is locked at OldCrusty Domains, it cannot be transferred. In general, this is a good thing. It prevents accidental transferring or deleting. How anyone would ever accidentally transfer a domain is beyond me, but hey, more safety never hurts.

However, to transfer the domain, we'll have to unlock it. Some registrars require us to call or email them to unlock a domain, but some are nice and give us a checkbox or setting in the control panel.

Configure WHOIS Information

If WHOIS Privacy is turned on, we also won't be able to transfer the domain. That's because an email will be sent to the **Administrative contact** listed in the domain's publicly available WHOIS information, and WHOIS Privacy blocks that email.

We'll also need to make sure the Administrative contact is an email that we can check. In the next step, OldCrusty will send a code to that email address, and we need that email. It's also a good idea to check if the WHOIS information is displaying correctly:

```
whois donkeyrentals.com

Domain Name: DONKEYRENTALS.COM
Admin Email: admin@donkeyrentals.com
...
```

Authorization Code

Now we need to request an authorization code from OldCrusty. Hopefully this is just a button we can push somewhere (if not, we may need to email or call OldCrusty support). An email will be sent from OldCrusty to the Administrative contact email address from our WHOIS information. As I said in the last section, WHOIS information needs to be publicly available (WHOIS Privacy turned off), and the Administrative contact email needs to be an email address we can access.

The authorization code, also called EPP (Extensible Provisioning Protocol) code, is a short code that we'll need to enter at ProbablyBetter domains. It looks something like `6sF>X95ZrH`.

Start the Transfer

At ProbablyBetter Domains we can start transferring in a domain. Every registrar is different at this point, but most likely, we'll enter our domain, agree to some terms of service, create or sign in to an account at ProbablyBetter domains, and enter that authorization code from the email. We'll also have to enter WHOIS information at ProbablyBetter Domains and enter payment information. This last step may come as a surprise.

The reason most transfers cost money is ICANN (the people who run the whole domain system) require the domain's expiration date be pushed out one year. So if the domain was going to expire on January 1, 1999, now it will expire January 1, 2000. For that reason, transferring often costs a similar amount to registering that same domain for the first time.

Minimize Downtime

Downtime is bad, and no one wants it, but it can be tricky to avoid when transferring a domain. Because of the way DNS works, servers often hold onto DNS data over long periods of time, even if we change our records.

This is normally not a problem because your DNS records don't change much. But when we transfer a domain we're essentially changing *all* the records. At any one point in time, DNS servers have records that are anywhere from 1 second to 48 hours old.

One way to work with this constraint, outlined here, is to create identical records at ProbablyBetter Domains and *then* transfer the domain over. That way if visitors access the new DNS records or outdated ones, it won't matter because they are the same.

This does require you to be able to create DNS records without transferring the domain first, which many registrars don't support.

Another way is to reduce the time servers hold onto old information. This is called the TTL value. It's the number of seconds before DNS servers refresh their data. The problem is, all servers that have your DNS data (potentially a lot) need to have this value updated also.

This answer on ServerFault shows the following steps:

> 1. Change the zone TTL to minimum - in most cases it's 300 seconds (5 minutes). Do not change any records at this stage.
> 2. Wait 48 hours.
> 3. [Transfer the domain]. It will take just 5 minutes to propagate the changes.
> 4. Revert TTL to standard 48 hours.

This is pretty clever. Most of the time, you want your TTL to be a long period of time so that if something *does* go wrong, it can't happen immediately to everyone. Also, this allows DNS servers not to have to be in near constant communication with each other, as that would be much harder to scale to the world-wide system that it is today.

Clean Up

After we've set up the transfer at ProbablyBetter Domains, we may need to accept that transfer back at OldCrusty Domains. Then, we wait. The process can take up to two weeks, so be prepared to wait that long. In my experience, it doesn't usually take that long, but we should give ourselves that much time just in case.

Also, know that our DNS records will not come over with the transfer. We'll need to re-add them manually if we haven't already. For small sites, this isn't too much of an issue, but if we have lots and lots of records, this could take awhile. Some sites let us set up the domain before we start the transfer. That can be a nice way to test everything.

Be sure to lock the domain and turn on WHOIS privacy if we want it after transferring.

Connecting a Domain to an External Service

There are tons of web services today that help us set up a website if we're not web developers. Gone are the days when we have to write our own blog, store, or company website, although some of us with masochistic tendencies tend to anyway.

These services are great, but they often give us a relatively non-personal URL, something like `our-name.service.com`. That works fine enough for testing, but running a serious business (such as renting donkeys) needs a professional and trustworthy domain name.

Sometimes, services will offer to sell us a domain up front. Obviously, domain experts like us don't need that kind of handholding. We want to be in control of our domains! What if we decide to switch services? Or have subdomains that point to other services? For me, this offering isn't flexible enough.

Connecting a domain that we own usually involves two steps: adding DNS records to point toward the service, then telling the service about the domain.

There are a few different ways to do the first step. The most common technique I've seen is to point an A or CNAME record toward an IP or hostname owned by the service. Sometimes, services force us to switch to their nameservers so they

can manage our DNS records. If this is the case, hope that they have good DNS controls if we ever want to set up other DNS records.

Next, we tell the service about our domain. Usually, in some control panel, we enter our domain so the service knows what domains to look for. This may take a bit to start working, but it's often immediate.

Check the Work

If it's not working, we can use `dig` to specify a nameserver. This returns 100% up-to-date information and can confirm that we set the records correctly:

```
dig @ns1.example.com donkeyrentals.com

...
```

Make sure you're querying the correct servers. If we're using the service's nameservers, we need to query those servers instead of our DNS provider's servers. That's where the updated records will be.

Also, be aware that if we want to use our bare, apex domain, we won't be able to use a CNAME record with the service unless our DNS provider supports ANAME/ALIAS. records. Some services force us to use a CNAME, in which case we'll have to use a subdomain like `www`. We'll explore solutions to this problem later.

Remove www From a Domain

Back in the 1990s, a convention was started: use a hostname to specify what service a domain represents. For example, an FTP server would put the `ftp` hostname on the front of their domain to create `ftp.example.com`, whereas email servers might use `mail.example.com`, and, of course, servers meant for the World Wide Web would use `www.example.com`.

Many web masters used this convention. It helped the public understand that `www.joespizza.com` on a takeout menu was a website, not a printing error.

But we are now decades beyond the 1990s, and the public is incredibly used to what a website is, yet the `www` remains widespread. There's no real harm in it, but we may want to get rid of it anyway. A more minimal URL can have a nicer aesthetic. Or perhaps we only purchased a TLS certificate for a single domain.

Whatever the case, it's possible to assure that visitors will be taken to the non-`www` URL regardless of what they typed in. The method we choose is dependent on our web server and whether or not we have access to its configuration files.

Note also that every method relies on a HTTP 301 redirect, which is the best way to signal to browsers and search engines that we'd really rather they not use the `www`.

nginx

If we use nginx and have access to its configuration files, removing the www is relatively easy. Find and open our configuration file, by default located at `/etc/nginx/conf.d/default.conf`.

There will already be a `server` directive for our main website. It should have these lines:

```
server {
  server_name donkeyrentals.com;

  # more configuration down this way...
}
```

This lets nginx know that when a request comes in for `donkeyrentals.com`, do something with it, ideally, serve up the website. Add this separate server directive above it:

```
server {
  server_name www.donkeyrentals.com;
  return 301 $scheme://donkeyrentals.com$request_uri;
}
```

We'll need to change `donkeyrentals.com` to our domain, of course.

This takes requests for `www.donkeyrentals.com` and, instead of serving up the website as normal, redirects (using a standard HTTP 301 redirect) to the non-www version of the site with the whole URL intact.

Apache

Apache's method is quite similar to nginx's: catch `www` URLs and redirect to the non-`www` versions. Open up our Apache configuration (usually found at `/etc/httpd/conf/httpd.conf`) and insert this directive at the top:

```
<VirtualHost *:80>
  ServerName www.donkeyrentals.com
  Redirect 301 / http://donkeyrentals.com/
</VirtualHost>
```

Then, replace `donkeyrentals.com` with our domain.

This, as we might expect, redirects `www` requests to the rest of the website using an HTTP 301 redirect. Apache preserves whatever is after the trailing `/` when using `Redirect`, so deep links will be maintained.

.htaccess

If we don't have access to our web server's configuration files, an `.htaccess` file can also remove the `www` prefix. Note that this only works if we have an Apache server running, which is very common, especially on shared hosting where we may not have direct configuration access.

Create a file called `.htaccess` in the root of our website's directory, i.e., on the same level as our `index.html` file. If one already exists, we can use that instead. Edit the file to have these lines:

```
RewriteEngine On
RewriteBase /
RewriteCond %{HTTP_HOST} !^donkeyrentals.com$ [NC]
RewriteRule ^(.*)$ http://donkeyrentals.com/$1 [L,R=301]
```

As usual, replace `donkeyrentals.com` with our domain name.

This looks for any URL that does not begin (`!^`) with `donkeyrentals.com` and redirects (again, using the HTTP 301 redirect) to the non-`www` website. This will only work for web requests, so other protocols like FTP will still work as planned. Other subdomains usually store their content in a separate directory so they won't be affected.

Use a CNAME on the Apex Domain

Occasionally, we'll need to use a CNAME record to redirect our domain elsewhere, but we'll want to use it on our apex domain, not `www` or another subdomain. The CNAME is usually the requirement, and the lack of a `www` is a stylistic choice. Some services (I'm looking at you Heroku) require that we use a CNAME record to point our domain toward their service.

We all know by this point that CNAMEs can't be used on the apex domain, so we're stuck using a subdomain. The common choice is certainly `www`, but, as previously established, maybe that's not what we want.

ALIAS/ANAME

If our DNS provider offers ALIAS or ANAME records, that's probably our best option. I'm not a huge fan of these because they are non-standard records, but sometimes they're our only choice.

On the upside, they are simple to configure:

- **Hostname**: `@`
- **Record Type**: `ALIAS`
- **Target Host**: `someotherwebsite.com`

And boom, `donkeyrentals.com` now points to another place.

Forwarding

Unfortunately, sometimes we don't have these special records and we have to use a subdomain like `www`. In this case, I prefer using forwarding to force all apex domains to redirect to their `www` counterparts. We still end up with `www` at the start of our domain, but people can visit either URL and get forwarded to the correct place.

In the last section, I went over detailed instructions on *removing* the `www` from a domain. We can use those instructions to go the other way and force the `www`. The two basic ways to do this are:

- Configure nginx or Apache to forward the domain.
- Use an .htaccess file to forward the domain.

Some DNS providers have tools to forward a domain, which works similarly to the methods above. In this case, we're forwarding it toward the `www` version. For example, if we have `donkeyrentals.com` as our domain, we can forward that to `http://www.donkeyrentals.com`. Then, our CNAME record for `www` can point wherever we need.

The Website Is Only a Blank or Placeholder Page

Getting a new domain is fun and all, but it can take some time to set up. A common problem is the website will show up as either nothing or some placeholder page.

Let's say we get a domain from Donkey Domains. With no setup whatsoever, domains often point to a placeholder page with a "website coming soon" message. There are two possible problems.

Default A Records

First, the A records for the domain haven't been configured yet and are pointing to their default IP addresses. This is an easy fix: point them toward our server. If we don't see the change, we can test our configuration with dig. We'll need a nameserver of the DNS provider. For this example, we'll use `ns1.donkeydomains.com`:

```
$ dig @ns1.donkeydomains.com donkeyrentals.com +short

12.34.56.78
```

Compare the IP address that comes back with the IP address of our server. If they're the same, it's likely we need to wait for the DNS to propagate throughout the internet.

Wrong Nameservers

If not, the second thing that could be wrong is our nameservers are set incorrectly. When our computer is looking up (*resolving*) the domain, the domain resolver will be pointed toward the wrong nameservers, then look in those servers for DNS records. If we remember from the *NS Records* section, this is like being in the wrong phonebook. If someone looks for us in a different city's phonebook, they won't find our number.

Some web hosts require us to use their nameservers instead of our DNS provider's. This is common when a domain comes with our web hosting. I like keeping my DNS records at the DNS provider because it feels nice to have them separated. Also, I suspect that most DNS providers do a better job at, well, managing DNS records than most web hosts. If DNS hosting is your core business, you're more likely to make sure it works well.

But sometimes we have no other choice but to host our DNS records at our web host. To do this, we'll need our web host's nameservers. They often look like `ns1.somedomain.com` or `ns2.somedomain.com`, or something similar, then it's generally as simple as finding the nameserver settings for our domain at the DNS provider. Commonly, these settings are not with our DNS records, but in some other, more general configuration spot.

Warning: When you change your nameservers, all of the DNS records at that nameserver become invalid. That's because any domain resolver will be looking for records on a different server now, so any old records at that old server will no longer work. You'll have to enter them all again at your web host.

My Old Website Is Showing Up

Let's say we've recently moved to to a different web host, or, for whatever reason, need to point our domain to a different server. This can be tricky because there's not a great way to test our records before we move. The best way I've found is to have a spare domain lying around that we can configure without fear of screwing anything up.

But that isn't always the case. So what can we do if it's not working?

Check the Nameservers

One very common problem is a domain's nameservers are set incorrectly. Remember, the nameservers are like the phonebook and all our other DNS records are like the people listed inside it. If someone is looking in the wrong "phonebook," they won't be able to find the right DNS records.

Some web hosts will manage a domain's individual DNS records, and the DNS provider is only responsible for administrative tasks like domain renewal and whois information. In a situation like this, we use the DNS provider to point our domain's NS records toward our web host, and the web host handles all other records.

Let's say we used Donkey Domains for our domain name and Horse Hosting to host our website. Over at Donkey Domains, we would set our NS records to the nameservers provided by Horse Hosting:

- **Nameserver 1**: `ns1.horsehosting.com`
- **Nameserver 2**: `ns2.horsehosting.com`
- **Nameserver 3**: `ns3.horsehosting.com`

Then, all other records (A, MX, CNAME, etc.) will be created through Horse Hosting's control panel.

Check the Records With dig

If we think we've set everything up correctly but our website is still not showing the correct data, we can check the records directly using `dig`.

First, we can check to make sure nameservers are what we expect them to be:

```
$ dig +short donkeyrentals.com NS

ns2.horse-hosting.com.
ns1.horse-hosting.com.
ns3.horse-hosting.com.
```

Then we can check individual records:

```
dig +short donkeyrentals.com A

104.131.191.2
```

We can also check the nameserver directly for any record:

```
$ dig +short @ns1.horse-hosting.com donkeyrentals.com A

104.131.191.2
```

If these match but we're not seeing the website in our browser, it's possible that DNS resolvers around the world haven't updated to the latest set of data yet. This is why we see warnings like "changes may take up to 48 hours."

However, if they don't match, that means we may have set the individual DNS records in the wrong place. Looking at the queries above, there are two types: one uses our DNS resolver, while the other queries the nameserver directly.

If the query using the nameserver is correct but the query using the resolver isn't, it's still possible that a resolver somewhere along the line is using old, cached information. This should sort itself out within hours or, at most, a couple of days.

However, if the direct nameserver query has the incorrect records, it's likely that the nameserver is still pointing toward the old domain host. Make sure the domain's NS records point toward the new server.

Redirect One Domain to Another

Sometimes we move on from a domain. Perhaps a company is rebranding, or we sold the domain for a tidy profit. Or maybe we've grown apart from the domain. The domain doesn't "get" us anymore and we've found a new, fresher, more happenin' domain.

Whatever the case, we want to make sure previous visitors to our domain can get to us at the new one, even if they visit the old one. Let's look at a few options on how to do this.

Our DNS Provider Might Do This for Us

If we're no longer using any part of the domain (e.g., no subdomains, email accounts, etc.), sometimes our domain host can redirect the entire domain for us. This is often called **forwarding** or **redirecting** the domain. We can check in the DNS provider's control panel for an option to forward or redirect.

If we have the option, the best thing to use is a 301 or permanent redirect. This will tell search engines and browsers alike to use the new domain instead of the old one. This technical detail may not be apparent in our control panel, but if we see it, we'll know we're doing it right.

If We Have Access to the Server

If we're using some part of our domain, but still want to redirect traffic, we can configure the web server to use a 301 redirect.

For an **nginx** server, we'll need to find the configuration file, by default located at `/etc/nginx/conf.d/default.conf`. Find the `server` directive for our website that has our old domain listed next to `server_name`, then change it to look like this:

```
server {
  server_name donkeyrentals.com;
  return 301 https://newdomain.com;
}
```

For **Apache**, find its configuration file, by default located at `/etc/httpd/conf/httpd.conf`. Find the `<VirtualHost>` directive that has our site listed next to `ServerName`. Add a `Redirect` line to it:

```
<VirtualHost *:80>
  ServerName donkeyrentals.com
  Redirect 301 / https://newdomain.com
</VirtualHost>
```

We can see these both use 301 redirects.

If we don't have access to the direct Apache configuration files, but we do have access to the files on our server, we can use an `.htaccess` file to redirect. Place or edit an `.htaccess` file in the same directory where our `index.html` page is. We may have to show invisible files if it doesn't show up.

Add this to the file:

```
Redirect 301 / https://newdomain.com
```

This looks very similar to our Apache configuration, and it does pretty much the same thing.

If We Don't Have Access to the Server

If the domain is the only thing we have access to, then the above is not an option. As long as we can create new DNS records, we can use a CNAME.

This is a book about DNS, so it might seem odd that I put the DNS solution last. That's because there's one big caveat: CNAMEs cannot exist on the apex domain. You might remember this from the Types of DNS Records chapter.

If we were using `www` or some other subdomain as our main URL, this will work great. Make a new CNAME record that looks like this:

- **Hostname**: `www`
- **Record Type**: `CNAME`

- **Target Host**: `newdomain.com`

Unfortunately, if our domain has no prefix, the only other option is an ANAME/Alias record, which our DNS host may or may not support.

Check Our Work

Obviously, we can visit the old website and see if the new one shows up instead. If we want to inspect further, we can use a command called `curl`. Open up a shell and type the following using the old domain:

```
$ curl -I donkeyrentals.com

HTTP/1.1 301 Moved Permanently
Location: httpw://newdomain.com
Date: Mon, 11 Apr 2016 17:51:57 GMT
Server: lighttpd/1.4.28
```

The `curl` command makes a request to whatever domain we tell it, like typing a URL into a web browser would do. The `-I` flag says that we want to see the HTTP headers that come back as a result of the request. In this case, we can see we got the `301 Moved Permanently` status code, which is exactly what we want. Go us!

Securing Your Website

TLS and SSL

You've probably seen that lock next to the URL in your web browser on certain websites. It means you're visiting a website that has proven its identity and established a secure connection with your computer. When that lock is present, the data to and from that site is encrypted. No one but you and the website are seeing the data being passing back and forth; this includes data such as passwords, credit card numbers, Social Security numbers, or health information.

If your data is not secure, it can be seen and stolen by malicious hackers. When I talk about data being "secure," I'm talking about two different concepts working together. First, and what you might be thinking, is *encryption*. Software can take any data and scramble it up so it is unreadable. That same software (with appropriate permissions, of course) can descramble that information. Second, and arguably more important, is *identity* or *authenticity*. The ability to de-scramble information should only be given to people we trust, and they can only be trusted if we know who they are. This is what makes the authenticity part of security so important.

TLS (Transport Layer Security), also referred to as SSL (Secure Sockets Layer), makes this encryption and authenticity possible. When it's set up for a website, it shows up as a lock or green badge next to a website's URL and signals to visitors that the website is safe. If visitors enter their private data into the secure website, snoopers won't be able to see it even as it's being transferred right in front of their eyes.

The real magic of TLS is that the entire process of proving authenticity and setting up encryption happens out in the open. Even as these digital criminals watch all

of the data going back and forth, it's still secure. I say "magic," but it's not. It's computer security, and we're going to look at how it works later on in this chapter.

Websites and servers aren't configured to use TLS by default. To do so there are some hurdles to jump through, namely proof and money. Getting that lock next to the URL requires paying for a certificate and verification that we own the domain that we're certifying. Don't let the word "money" intimidate you, though, since free alternatives are starting to pop up.

These roadblocks exist because, as we talked about above, security entails some kind of identity or authenticity. For example, you trust the teller at your bank because you trust the bank owners, and they trust their employees. But if the teller came up to you on the street and said, "HEY, I CAN STORE YOUR MONEY!", you'd probably pretend not to hear them and walk away quickly.

That's the importance of authenticity, but what about encryption? How could data go right in front of hackers without them seeing it? Well, exactly *how* someone could hack isn't a topic for this book, but we can look at a simplified example of what they would see.

It happens to all of us: You're out in a public place like a coffee shop, and you just *have* to buy that artisanal, small-batch donkey saddle for your niece's birthday party. Someone in a trench coat and sunglasses with a laptop is sitting nearby watching all the Wi-Fi data go back and forth. They might see something like*:

```
...
User visited donkeyrentals.com
User visited checkout page
User submitted an order with this data:
    Name: Edward A Loveall
    Credit Card Number: 1234-5678-9012-3456
    Expiration Date: 12/99
    CVV: 123
...
```

**This was only a reenactment. No real data was stolen in the making of this example.*

Hackers can see this data flying by and record it. Now they have your credit card number with verification code and everything. No good! In contrast, if you have a secure connection established with TLS, the hacker would see something like this instead:

```
...
DeXZPuvv5btpckqk1feXpmUmLBuCMOrIboeg3WHs1rV8eydrTYVgDDq91u02O9HNijDNo+U
Y01IprNqHu6RAE+Z2vBq7jXgKAOqiHsq71yOnfZKiXJThi5u24kRnh1Rm9zht5lNzQ87yI9
KGPf8p16gplQ1SfTa85LLZWkglZGhfnHkVWNdgPf5rpQIWby24YhTPswG4ZSXw4S/pJKhms
6trB2gSCxJi4zJUPSCmB24V3HpdIL1ZPEIRwAz4EpYir/BEefVenOT8pW6afkyp6wKxInNz
pB/16OE0qzjYYzNXymdIAmlzoglLEMIX1bMpZPTPfs2gLJdCr1oQCJ5z1gfXPL2veyK1PaO
...
```

This is ever-so-slightly less readable. Most nefarious hackers would see this and move on to someone else, but even if they decided to grab this snippet, it would be nearly impossible to decode, and it's not even guaranteed to be useful. It might just be someone merely *browsing* donkey saddles.

Most of this chapter is about obtaining one of these certificates. In addition, there are some other related topics we'll cover:

- What certificates are and their different types
- Certificate authorities and how they relate to certificates
- Information we'll need to correctly set up a certificate
- Generating a private key and certificate signing request
- Obtaining a certificate from a certificate authority
- Installing our certificates with nginx or Apache

Let's get started!

Fair warning: This is a dense chapter that you may have to read more than once. Make sure you've had a full night's sleep and waited 30 minutes after eating. We're about to go into the deep end.

Certificates

You can think of certificates like your driver's license: all your identifying information bundled up into a document. It has everything that proves that you are

authentically... well, *you*. When you want to drive or drink legally (not at the same time!), you are asked for your information. But if you only claimed, "Hey, I'm 21 years old and a citizen of this country", that wouldn't convince anyone. You need that license.

To get a license, you need to go to a government building for issuing licenses and fill out some forms. Now you have something that A) identifies you and B) is trusted by others. The government trusts the people in this building and the building trusts you. This is called a **chain of trust**. This also allows the wider world (or at least country) to trust you when you show them your license.

In this analogy, your license is your certificate. It's stamped by the government (which represents the certificate being created) because you filled out the right paperwork for a license (i.e., a certificate signing request).

Certificate Authorities

Also abbreviated CA, this is where we'll buy our TLS (also called SSL) certificate from. All CAs have a public certificate which can be used to verify that pieces of information have come from them. In fact, visitors to our site or service will use them to verify our certificate. More on that later.

Some CAs are called "root certificate authorities," so we call their certificates "root certificates". All operating systems these days come with a bunch of these root certificates, which start the chain of trust I mentioned earlier. To see a list of root CAs installed on your computer, search the web for `<operating system> trusted root certificates` and insert your operating system. You'll find a long list of certificates that all have very enterpise-y sounding names like *UltraCorp Secure* or *DonkeyCom Gold*.

Public-key Infrastructure

This whole network of certificates, certificate authorities, and the chain of trust uses a system called public-key infrastructure or PKI.

It starts with a **public/private key pair**, which are two files (public key and private key) that are "cryptographically matched." This means that a public key can encrypt data that *only* the private key can decrypt. Also, because they are matched, if either

key is tampered with even just a tiny bit, they become completely unmatched and will be unable to encrypt/decrypt for each other. We will be making a key pair for our server. All CAs already have one.

Public keys, as their name suggests, are public. In fact, whatever CA we choose will have a public key that anyone can see. All CAs do this, and we will too. Private keys, on the other hand, are the exact opposite. They need to be kept 100% secret and secure. I'll be harping on this for the rest of the chapter: **private keys stay private!**

When we want a certificate from a CA, we generate a key pair. We send them the public key along with some other information about us and our domain name. They bundle that up and send it back as a certificate. It contains our public key, a **signature** from the CA, and meta-information such as what CA it came from, how it was created, and more.

That signature is a new piece of this puzzle. The CA creates one by first taking our public key and turning it into a **digest**. A digest is made by transforming (or hashing) some input into some output of a standard length. Think of it as a fingerprint of our public key: a much smaller, identifying version of us that is the same every time. Second, the CA makes the signature by combining this digest along with their **private key**.

By the way, this private key from the CA is just like the private key we will create. It's kept secret *just like you should keep yours secret* and has an associated public key. The CA's public key is the same public key that is stored on all major operating systems.

This all comes together when someone visits our secure website (or server) and the browser (or client) downloads the certificate we've set up. That certificate contains our **public key**, the **signature**, and some other meta-information, such as who created it and what information the certificate is certifying. In our case, we're certifying our domain name.

The browser then takes our public key from inside the certificate and creates a digest, just like the CA did. Next, it looks to see which certificate authority issued the certificate and grabs that public key from the operating system.

It then uses the CA's public key, the digest, and the signature from the certificate to verify that the public key is A) ours and B) hasn't been tampered with: **certificate authority's public key** + **digest** + **signature** = **verification**. If the verification

comes back as okay, it proves that we are who we say we are, and we can now be trusted by the browser!

After that, the server and the browser set up a secure connection (we'll talk about this connection a little later), and all data from there on out is encrypted. *Phew!*

Different Levels of Validation

There are three different levels of validation. These levels are representations of how much you have proven to a certificate authority. The first is a simple **domain validated** (or **DV**) certificate. This is by far the simplest and only requires that you prove you have control over a domain. Proof can be obtained when the CA sends an email to an address listed in the domain's WHOIS information or sends an email to a common email address like `admin@yourdomain.com` or `webmaster@yourdomain.com`.

If we don't have email set up for that domain or WHOIS privacy is turned on, there will be no email addresses to send to. In that case, there are usually other ways to validate the domain. For example:

- Put a text file with a very specific name on our web server for that domain. The CA will then check for this text file.
- Create a DNS record like a CNAME or TXT record that the CA can check for.

Both of these low-hassle methods prove that we own the domain.

More complicated but more trustworthy for visitors than a DV certificate is an **organization validated** (or **OV**) certificate. This requires legal proof that your organization exists, such as:

- The business's address in an official government database or other third-party database
- Articles of incorporation, business license, company bank statement, and other similar documents
- The driver's license or passport of the applicant
- A recent major utility bill

This can vary between certificate authorities but, as you can tell, it's much more involved. Interestingly, these certificates are actually somewhat hard to find. You most often find DV's or their more robust sibling:

Extended validated (**EV**) certificates are the most rigorous of all, requiring many levels of proof. For example, Comodo requires the proof of:

- Legal existence and identity
- Trade/Assumed Name as applicable
- Operational existence
- Physical address and organization phone number
- The name, title, authority, and signature of the person(s) involved in requesting the certificate and agreeing to the terms and conditions

While this kind of certificate is the most effort, it is rewarded with a special badge in the URL bar. Go to a site like GitHub and see the green badge next to the URL. That is an extended validated certificate in action. Compare that with a site like Wikipedia, which only has a domain validated certificate. Most browsers will let you click on that lock or badge and see the certificate details and the chain of trust. That chain leads all the way back up to the root certificate, which we mentioned above.

The advantages of OV or EV certificates over a DV are unclear. There's certainly no harm in having more verification, other than the extra time, effort, and money it takes to set up. You can expect to pay about 5-10x more money for an EV, and it can take several weeks to be verified. There's no extra encryption that comes with them either. From a passing-around-data perspective, they are all equally secure.

One rationale for OV or EV certificates is this: "Customers will trust a website with a larger, greener, more official-looking badge." In this writer's opinion, most customers don't even notice. Those that do notice are not likely to look into the legitimacy of the certificate that much. My advice is start with a DV. It will get you up and running with a secure website faster, which is a legitimate benefit. If you need to upgrade to an OV or EV later, the option's always open.

Wildcard

One restriction to a plain DV certificate, however, is we can only encrypt a single domain. We pick something like `donkeyrentals.com` and that's it. If we need to

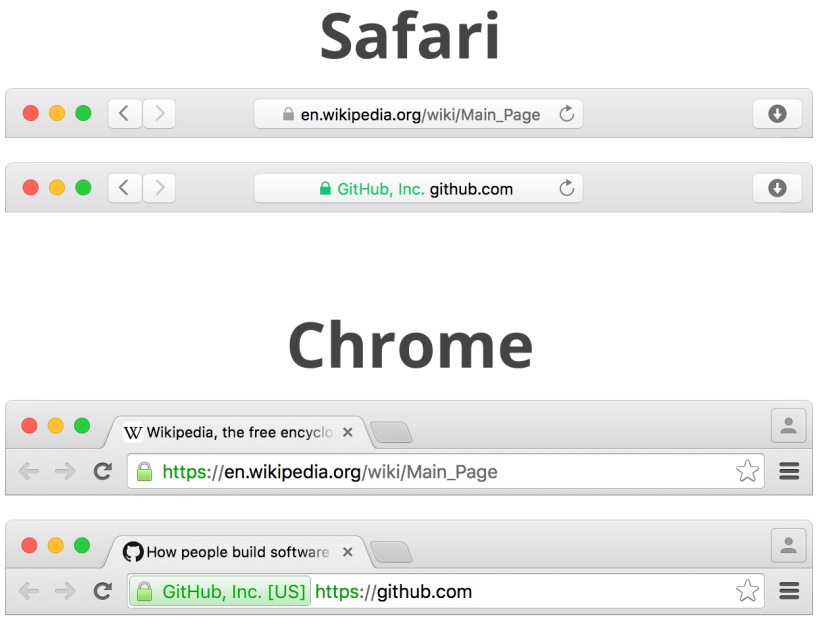


Figure 6.1: TLS certificate badges in the Safari and Chrome browsers

secure subdomains, we can get a wildcard certificate.

This lets us secure `*.donkeyrentals.com`. That is, any domain attached to `donkeyrentals.com` such as `blog.donkeyrentals.com` or `shop.donkeyrentals.com` can now be secure too. Note, however, that we can't secure `secret.blog.donkeyrentals.com`. Only one level of subdomains is permitted. Wildcard certificates usually cost about 2-4x more than a regular DV, but if multiple subdomains enter the picture, it can often pay for itself.

Be Prepared

So what do we need before we setup a TLS certificate on our website?

A Certificate Authority

We talked about this in the previous section, but this is who you will buy your certificate from. There are many certificate authorities to choose from; searching for `buy SSL certificate` will find plenty. Some to check out are:

- Comodo
- DigiCert
- RapidSSL (one of the least expensive paid CAs I've found)

And one that offers free DV certs:

- Let's Encrypt

One thing you'll notice when shopping around is paid certificates are often much more expensive than domain names. For example, the certificate for donkeyrentals.com cost $230.85 for three years, or $76.95 per year. The domain cost $13.17 for one year. Security can be expensive.

That said, there are free alternatives like Let's Encrypt if you only need a DV certificate. "How is Let's Encrypt free?" I hear you asking. Well, they are funded by a

bunch of tech companies to promote the adoption of digital certificates. Let's Encrypt overall looks really great and I've heard many system administrators praise it. Check it out if you're looking for a free option.

For this chapter we're going to be looking at more traditional certificates that are purchased and installed manually.

OpenSSL

OpenSSL is software that implements these TLS standards we've been talking about. OpenSSL has a ton of functionality, and we're going to use it to generate our server keys, create our certificate signing request (CSR), and debug our certificate.

Make sure the `openssl` tool is installed before continuing. It's often (but not always) installed on Unix/Linux systems. You'll probably have to install or update it on Windows and Mac OS X. Open up a terminal or command line and type `openssl version`. If you get something like `OpenSSL 1.0.2g  1 Mar 2016`, you're all set. If not, you'll need to install it. If you have an old version, I'd recommend updating it.

Also, I'll note here that OpenSSL has been the cause of some security bugs in the last couple of years. Since it is a very popular implementation of TLS standards, any bug found is able to be exploited across a large number of websites. You may have heard of heartbleed, a bug that allow hackers to access server data, including the server's *private key*. Like any piece of software, this is one of many bugs that happen in the course of development. Unlike any piece of software, bugs in OpenSSL are more extreme because they can jeopardize server and customer security.

Because of these bugs, other companies have started to make their own implementations:

- Mozilla has created NSS.
- Amazon is taking a stab at modernizing OpenSSL with s2n.
- OpenBSD created LibreSSL.
- Google created a version called BoringSSL, but they don't recommended it for public use.

If you'd rather use these, then great! However, for the rest of this book I'm going to be using OpenSSL commands. Make sure you translate any commands as necessary if you're going to follow along with a different tool.

A Common Name

This is the full domain that we'll be securing. I say "full" because, for a certificate, just as in DNS, `www.donkeyrentals.com` and `donkeyrentals.com` are different. If you're purchasing a wildcard certificate, this will probably be the equivalent of `*.donkeyrentals.com` to cover all first level subdomains. For a single DV certificate, you'll have to pick your apex domain or a subdomain. This could be `www.donkeyrentals.com`, or `secure.donkeyrentals.com`, or `donkeyrentals.com`, etc. You can't change this without getting another certificate, so choose wisely.

A Web Server

This might seem obvious, but the web server that you use matters quite a bit. If you're setting a web server up manually, it's likely that you're using Apache, nginx, or IIS. Any web server worth using supports TLS certificates. You'll also need access to the configuration files of these web servers.

If you're using shared hosting (Dreamhost, Bluehost, etc.) or a service (Tumblr, Squarespace, Shopify, etc.) to host your website, check with that service to make sure they support TLS/SSL and allow you to configure enough of your web server to set up a certificate. Some do, some don't. Some have restrictions on the kinds of certificates you can use or how you use them. Some only let you configure a small part of your web server. I can't give any more advice here than to contact the service or do some very extensive research to find out what the restrictions might be.

Getting a Certificate

Let's walk through creating a private key and certificate signing request (CSR) step by step. We'll upload the CSR to our Certificate Authority (CA) and they will create and sign our certificate. After, we'll cover installing that certificate in a couple of places. By the end of the chapter, we should have ourselves a secure website.

Create a Working Directory

We're going to be creating a few files, so it's nice to have them contained in a single folder. This folder should eventually be secured, because it will contain private key information that should *never* be made public. Create a directory with a name like `donkeyrentals-certs` or something equally descriptive. Navigate to that directory inside of your shell.

Create the Private Key and CSR

Next, we're going to create our private key and CSR. We'll need to enter a bunch of contact information for this. It can be a little overwhelming, so let's do one together. If you messed up, starting over is as easy as deleting the two files it creates and running the command again. I'll run through Donkey Rentals, and you can follow along with your domain:

```
$ openssl req -nodes -newkey rsa:2048 -keyout donkeyrentals_com.key \
-out donkeyrentals_com.csr

Generating a 2048 bit RSA private key
.....................................+++
.....................................+++
writing new private key to 'donkeyrentals_com.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:Massachusetts
Locality Name (eg, city) []:Cambridge
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Donkey Rentals
Organizational Unit Name (eg, section) []:.
Common Name (e.g. server FQDN or YOUR name) []:donkeyrentals.com
```

```
Email Address []:.

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:crouching-domains-hidden-donkeys
An optional company name []:Donkey Rentals
```

We'll go over all those fields in a second. First, we can look at the two files created by the command:

- donkeyrentals_com.key

The private key. Again, this is the thing we need to keep absolutely secret.

- donkeyrentals_com.csr

The certificate signing request. We will be uploading this to our CA. Think of it as the digital paperwork you have to fill out and submit to get your certificate. It contains the public key and the contact information we filled out above.

These names aren't hardcoded; we entered them in the command above. While it doesn't matter what we name the files, it's a good idea to name them in a way that will give us a hint as to what they will be in the future. A convention is naming them similar to the common name, e.g., the `donkeyrentals.com` CSR is named `donkeyrentals_com.csr`.

Let's briefly go over the fields we filled out when creating the key and CSR. Most of these fields should be self-explanatory, but if you're easily intimidated like I am, this should help. To leave a field blank, use a single period (`.`) character.

- **Country Name** – The two-letter country abbreviation for the certificate's owner.
- **State or Province Name** – Use the full name here. If you are in a country that doesn't have these divisions, use the full country name instead.
- **Locality Name** – The City, Township, Hamlet, Ant Colony, etc.
- **Organization Name** – The company representing the certificate.
- **Organizational Unit Name** – You can leave this blank (with a period).

- **Common Name** – We talked about this back in the *Be Prepared* section. This is the exact domain we want to encrypt. `donkeyrentals.com` vs `www.donkeyrentals.com` is crucial here.
- **Email Address** – This one can stay blank, too (again, with a period).
- **A challenge password** – Make this a very strong password, unlike the one I used in the example.
- **An optional company name** – This can be the same as **Organization Name**.

And that's it for CSR.

Upload the CSR

The next thing to do is provide the CSR to our CA. It's a text file that looks like a bunch of gibberish with a header and footer. It contains your public key and all the information we just typed in. Your CA will know how to read it. In some cases, we need to upload the .csr file, but in others we resort to copy-and-paste like animals. Your CA will tell you what they want you to do.

Either way, once we get our CA to sign our certificate, they'll send it to us, probably in an email, or make it available on their website. It also may come with a `ca-bundle` file that we will use with the certificate. This usually happens within minutes for a single domain certificate. Once we have the certificate, let's save it to our working directory as `donkeyrentals_com.crt` or your equivalent file name.

Now it's time to install it. If you don't manage your own web server, you'll have to rely on your web hosting provider to provide instructions. If you use nginx or Apache on a server, let's dive in.

Installing a Certificate for nginx

What We Need

First, we'll need to be able to access our server. This is usually done through SSH, which I will assume you have set up already. Also, make sure that we have the certificate files we got from our CA:

- `donkeyrentals_com.crt`
- `donkeyrentals_com.ca-bundle`
- `donkeyrentals_com.key`

Finally, we'll need to install `nginx` and `openssl`. To find out if nginx is already installed, we connect to the server and type:

```
nginx -v

nginx version: nginx/1.6.3
```

To see if openssl is installed:

```
$ openssl version

OpenSSL 1.0.2g  1 Mar 2016
```

If we get something back that says `command not found`, then the package is not installed. If that's the case, our package manager (yum, brew, apt, choco, etc.) should be able to install them for us.

Combine the Certificates

nginx needs the certificates combined into one file. The order of the files should be:

```
Our certificate
Intermediate certificates
Root certificate
```

Our certificate authority should give you instructions on how to do this and the exact order they want the certificates in. They will (or *should*) also provide us with any certificates we don't have. We're essentially creating a file that starts with our certificate and tracing a path back to the root certificate, via our CA.

We can combine them on our local computer. For example:

```
$ cat donkeyrentals_com.crt donkeyrentals_com.ca-bundle > ssl-bundle.crt
```

Now we have one big certificate file (`ssl-bundle.crt`) that contains our certificate, any intermediate certificates, and, finally, the root certificate.

Upload Those Certificates and Key

Next, we need to put the files on our server. I like to put them in the `/etc/ssl` directory. The certificate bundle we just created goes in `/etc/ssl/certs/ssl-bundle.crt`, and the private key goes in `/etc/ssl/private/donkeyrentals_com.key`. These don't have to go here specifically, but it's a good convention for nginx, and I'll assume they are in these directories for the rest of the tutorial.

Also, make sure our `/etc/ssl/private` directory has good permissions. Only the root user should be able to access it:

```
$ chmod 700 /etc/ssl/private/
$ chown root /etc/ssl/private/
$ chgrp root /etc/ssl/private/
```

Configure nginx

We can make a new file in our nginx config directory for our website. Mine is at `/etc/nginx/conf.d/donkeyrentals.conf`. Here is the file in full:

```
server {
  listen 443;
  server_name donkeyrentals.com;
  root /var/www/donkeyrentals;

  ssl on;
  ssl_certificate /etc/ssl/certs/ssl-bundle.crt;
  ssl_certificate_key /etc/ssl/private/donkeyrentals_com.key;
  ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
  ssl_ciphers "EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH";
  ssl_prefer_server_ciphers on;
}
```

Let's walk through each line:

- `listen 443;`

The server should listen for connections on port 443. This is the conventional port for TLS connections. When we visit a site at `https://` as opposed to `http://`, it will try to make a request using port 443 instead of the standard port 80.

- `server_name donkeyrentals.com;`

When requests come in for `donkeyrentals.com` to our server (on port 443), they will now be dealt with instead of thrown away. This must match the common name we used to create the CSR earlier.

- `root /var/www/donkeyrentals;`

The location of our html (and other) files for donkeyrentals.com on this server.

- `ssl on;`

Enable TLS/SSL.

- `ssl_certificate /etc/ssl/certs/ssl-bundle.crt;`
- `ssl_certificate_key /etc/ssl/private/donkeyrentals_com.key;`

The locations of our `ssl-bundle.crt` and `donkeyrentals_com.key` files. Again, these are what I consider to be sensible locations for these files, but they can technically go anywhere. Keep that `.key` private like your donkey-renting life depends on it.

- `ssl_protocols TLSv1 TLSv1.1 TLSv1.2;`

Enable only the newer TLS protocols. This implicitly disables old, outdated, and vulnerable versions of SSL so they can't be exploited by hackers.

- `ssl_ciphers "EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH";`

Set the ciphers that the client is allowed to use for encryption. There are many different ciphers used to encrypt data when we have a secure connection. The list above is one currently recommended set of ciphers. A full list of ciphers can be seen here.

Side note: picking which cypher suites to use is tricky. It depends on our server, the client, the current state of security, and many other factors. If you want to learn more, there are lots of resources that have more information.

- `ssl_prefer_server_ciphers on;`

Use the server's encryption algorithms instead of the client's when using TLS.

Final Steps

Really all that's left is to reload nginx and check out our server. Visit `https://donkeyrentals.com` (note the **s**), and we should see a little lock next to the URL. If so, nice work! If not, well, time to go back and read all those instructions again. A day in the life of a server administrator.

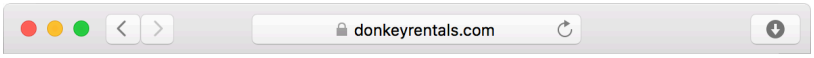


Figure 6.2:

Installing a Certificate for Apache

What We Need

This might be a bit obvious, but we'll need access to our server. We'll need to have Apache installed (sometimes referred to only as `httpd`). And, of course, we need the certificate files that our CA sent us:

- `donkeyrentals_com.crt`
- `donkeyrentals_com.ca-bundle`
- `donkeyrentals_com.key`

Additionally, we'll need to have `mod_ssl` and `openssl` installed. We can check for `mod_ssl` by running one of these two commands, depending on which version of Apache we're running:

```
# Apache 2
$ apache2ctl -M | grep ssl

# Apache 1
$ apachectl -M | grep ssl
```

If `mod_ssl` is installed, we'll see something like:

```
ssl_module (shared)
```

Checking `openssl` is straightforward:

```
$ openssl version

OpenSSL 1.0.2g  1 Mar 2016
```

If we get something back like `openssl: command not found`, then we don't have it installed. If these are missing, our package manager (yum, brew, apt, choco, etc.) should be able to install them for us.

Finally, you'll need to know where your Apache config files are. They are commonly in the `/etc/httpd` directory. If they're not there, a good way to find them on Linux systems is the following command:

```
$ sudo find / -name httpd.conf
```

This will locate the main configuration file, `httpd.conf`. From there, we can probably find all the other directories we need like `modules`, `conf.d`, etc. We'll need these directories later.

Move All the Files

Now that we have all the files we need, we can place them in some sensible directories. If the directories don't exist, we should create them. The `.crt` and `.ca-bundle` files can go in the `/etc/ssl/ssl.crt` directory, and the `.key` file will go in the `/etc/ssl/ssl.key` directory.

We'll also set good permissions on the private key's directory. We want only the root to be able to access the file so no one else can see our private key.

```
$ chmod 700 /etc/ssl/ssl.key/
$ chown root /etc/ssl/ssl.key/
$ chgrp root /etc/ssl/ssl.key/
```

Configure Apache

Next, we need to tell Apache where to find these files and how to act as a secure server. It's possible a configuration may already exist that we can change. Let's search inside our config files to check:

```
$ grep -r SSLCertificate /etc/httpd
```

This found the file `/etc/httpd/conf.d/ssl.conf`, which we can modify to our liking. If no file was found, we can build a configuration up from scratch.

A pre-existing file will have hundreds of lines, many of them commented with a `#` at the start of the line. Here's a no-comments version of the relevant part of an example configuration:

```
LoadModule ssl_module modules/mod_ssl.so

<VirtualHost *:443>
DocumentRoot /var/www/donkeyrentals
ServerName donkeyrentals.com
SSLEngine on
SSLCertificateFile /etc/ssl/ssl.crt/donkeyrentals_com.crt
SSLCertificateChainFile /etc/ssl/ssl.crt/donkeyrentals_com.ca-bundle
```

```
SSLCertificateKeyFile /etc/ssl/ssl.key/donkeyrentals_com.key
SSLProtocol -all +TLSv1
SSLCipherSuite EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH
</VirtualHost>
```

Many of these lines might already exist and need no modification. Some only need to be uncommented from the existing file and tweaked slightly. Some will need to be added entirely. Order doesn't matter too much, as long as all the lines are inside of `<VirtualHost *:443>` and `</VirtualHost>` (except for the first `LoadModule` line). Here's what each line does:

- `LoadModule ssl_module modules/mod_ssl.so`

This tells Apache to load the `mod_ssl` package we installed earlier. The `modules/mod_ssl.so` path is important here. It's relative to the `ServerRoot` directory, which is by default `/etc/httpd`. The full path in the example would be `/etc/httpd/modules/mod_ssl.so`.

- `<VirtualHost *:443>`

We start the `<VirtualHost>` directive off by saying this applies to any IP address, `*`, on port `443`, which is the conventional port for secure connections.

- `DocumentRoot /var/www/donkeyrentals`

The location on this server where we can find our website content like the `index.html` file, images, scripts, etc. is:

- `ServerName donkeyrentals.com`

The name that the server identifies itself as, i.e., our common name, is the same thing we entered when making our CSR.

- `SSLEngine on`

Enable TLS/SSL.

- `SSLCertificateFile /etc/ssl/ssl.crt/donkeyrentals_com.crt`
- `SSLCertificateChainFile /etc/ssl/ssl.crt/donkeyrentals_com.ca-bundle`
- `SSLCertificateKeyFile /etc/ssl/ssl.key/donkeyrentals_com.key`

Where to find our certificate, certificate authority bundle, and private key (these are the same files we moved around previously):

- `SSLProtocol -all +TLSv1`
- `SSLCipherSuite EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH`

Only use TLS version 1 protocol and higher, and use currently recommended cipher suites. A full list of ciphers can be seen here. Fancy stuff.

Side note: Picking which cypher suites to use is tricky. It depends on our server, the client, the current state of security, and many other factors. If you want to learn more, there are lots of resources that have more information.

- `</VirtualHost>`

Finally we make sure to close our `<VirtualHost>` directive.

If you need further customization, check out Apache's documentation on the SSL module.

Check Our Work

Now that we're configured properly, we can make sure we didn't cause any errors in our Apache configs with one of two commands:

```
# Apache 2
$ apache2ctl configtest

# Apache 1
$ apachectl configtest
```

If we see `Syntax OK`, then we're good to go.

Finally, we restart Apache:

```
$ sudo service httpd restart
```

You've now configured Apache to use your brand new TLS certificate. High five!

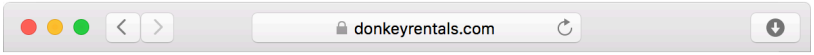


Figure 6.3:

Wrapping Up

Time to update your résumé, LinkedIn profile, and MySpace page because now you can make your website secure! You sat through chapters about DNS records, technical domain name details, and my bad jokes, so of course you were able to master TLS and the bevy of acronyms found in this chapter.

So what's next, my donkey-renting pupil? Well, you can always learn more about security. OWASP maintains a ton of information on current security threats, initiatives, and many other resources. SSLLabs has some great tools and guides to help ensure your websites are secure and follow the best practices. They even have a tool that will test and grade the security of any website.

A large part of security is keeping up to date with new patches and upgrades for the software you use. If you're using OpenSSL, I suggest following their security vulnerabilities.

The W3C has lots of great information about web security protocols and standards. Finally, if you're a regular to Hacker News, any major security news inevitably pops up there.

Another good step is finding out what your operating system recommends for security. Any system worth using will have security updates, guides, and best practices. A quick search for `<your operating system> security` should return quite a

bit.

Finally, as I noted previously, some people have been switching away from OpenSSL because of its shortcomings. If that sounds like your thing, here's that list again:

- NSS by Mozilla
- s2n by Amazon
- LibreSSL by OpenBSD
- BoringSSL by Google (not recommended for public use)

Glossary

A Record

An A record or *address record* is a DNS record type that points a hostname to an *IPv4* address. Example:

- **Hostname**: `www`
- **IP Address**: `12.34.56.78`

See the section on A records to learn more.

AAAA Record

An AAAA record or *quad-A record* is a DNS record type that points a hostname to an *IPv6* address. Example:

- **Hostname**: `www`
- **IP Address**: `2620:0:861:ED1A::1`

Note that this IP address example is shortened. See the section on AAAA records to learn more about shortened IPv6 addresses.

ALIAS Record

A non-standard record type first used at DNSimple. It allows CNAME-like redirections on the apex domain, which is not normally allowed. For example, `donkeyrentals.com` could point to `myapp.herokuapp.com` instead of an IP address. Normally, this would need to be done with `www.donkeyrentals.com`.

See the section on ALIAS or ANAME records for more information.

ANAME Record

See *ALIAS Record*.

Apex Domain

The domain without anything before it. Example: `donkeyrentals.com`, but not `www.donkeyrentals.com` or `magical.donkeyrentals.com`.

Bare Domain

See *Apex Domain*.

BIND

Short for *Berkeley Internet Name Domain*, this suite of software is made to run DNS servers. It is widely used across the industry. The BIND software suite includes tools used in this book, such as `dig`, `host`, `nslookup`, and many others. More information can be found on BIND's website: `https://www.isc.org/downloads/bind/`

ccTLD

Country Code Top-level Domains are domains representing a country. Examples include `io` (Indian Ocean), `bz` (Belize), and `fi` (Finland). Some ccTLDs have prerequisites for registration, but, for the most part, anyone can register a domain at any ccTLD.

See also: *TLD*.

Certificate Authority (CA)

Companies that issue certificates. This is where you go to upload a certificate signing request (CSR) and get back a certificate. Examples of these are Comodo, DigiCert, and RapidSSL.

See the section on certificate authorities for more information.

CNAME Record

Canonical Name records or *CNAME records* are a DNS record type that point a hostname to a domain name, as opposed to an IP address. Example:

- **Hostname**: `redirect`
- **Target Host**: `rabbitrentals.com`

Note: These cannot be used on the Apex Domain. In the example above, this would point `redirect.donkeyrentals.com` to `rabbitrentals.com`. You cannot use a CNAME to point `donkeyrentals.com` in the same way.

See the section on CNAME records for more information.

Common Name

In regards to TLS certificates, the common name (or CN) is the "place" that the certificate is securing. For example, a certificate for `donkeyrentals.com` will have `donkeyrentals.com` as its common name. The domain being visited and the common name much match exactly. If they don't, it will cause a security warning when a user visits the site.

Wildcard certificates are used if multiple subdomains need to be secure. In this case, the common name can use the `*` character to specify all subdomains, e.g., `*.donkeyrentals.com`. Now, users can visit any subdomain such as `secure.donkeyrentals.com` or `www.donkeyrentals.com` and the certificate will be valid.

dig

An abbreviation for *domain information groper*, this is a utility used to ask questions and receive answers from DNS servers about their records. For example, we can see the value of the A record for `donkeyrentals.com` with the following command:

```
$ dig donkeyrentals.com A
```

Dig is included in the *BIND* suite of software DNS utilities. It is similar to the *host* tool, but is far more verbose and explicit.

See also: *BIND*, *host*, *nslookup*.

See the section on dig for more information.

DNS

Short for (the) Domain Name System, the whole network of servers holding records that point to other records or servers. See the rest of this book for more information.

Domain Name Front Running

Often, registrar websites will let you search to see if a domain is already taken. You type in the domain name and the registrar says, "Taken, sorry." or "It's available! Would you like to register?" If you choose not to register, this domain name should just continue being unregistered until a customer comes along and grabs it.

But some malicious registrars will actually register the domain after it's been searched for, then sell it back to you for a higher price. The nerve! This is domain name front running.

FQDN

A fully qualified domain name, which contains all *hostnames* (including the root zone) combined and separated with a period. Since the root zone does not have any text associated with it like `com`, a FQDN ends with a period. Example:

```
shop.donkeyrentals.com.
```

gTLD

Generic Top-level Domains are, for the most part, domains that are not associated with a country. Examples include `com` (commercial), `org` (organizations), and `dentist` (dental practices). Some gTLDs have prerequisites for registration, but, for the most part, anyone can register any gTLD.

See also: *TLD*.

host

Host is a tool in the *BIND* suite of software used to ask questions and receive answers from DNS servers about their records. For example, we can see the value of the A record for `donkeyrentals.com` with the following command:

```
host -t A donkeyrentals.com
```

It is similar to *dig* but attempts to have a much more succinct and readable output.

See also: *BIND*, *dig*, *nslookup*.

See the section on host for more information.

Hostname

Each individual part of a domain. For example, in `food.donkeyrentals.com`, the hostnames are `food`, `donkeyrentals`, and `com`. Putting these together with dots in between creates a domain.

ICANN

The Internet Corporation for Assigned Names and Numbers plays a large part of running the internet as a whole. They run the root name servers where all of the TLDs are found. They also police misbehaving registries, decide what TLDs are allowed to be added, and assign IPv4 and IPv6 addresses.

InterNIC

This stands for Network Information Center, with the Inter- prefix presumably for the internet. In the early days of the internet, (1972–1990ish) they handled domain registrations for `com`, `edu`, `gov`, `mil`, `net`, `org`, and `us` domains. InterNIC is now part of ICANN.

IPv4

Short for Internet Protocol version 4, this protocol defines addresses as a sequence of four integers (0–255) separated by periods. Examples include `1.2.3.4`, `127.0.0.1`, and `273.115.5.53`.

IPv6

Short for Internet Protocol version 6, this protocol defines addresses as a sequence of eight groups of four hexadecimal digits (0–FFFF), separated by colons. Examples include `2620:0000:0861:ED1A:0000:0000:0000:0001` and `2620:0:861:ED1A::1`.

Some IPv6 addresses can also be shortened significantly, as seen in the second example above. See the AAAA records section to learn more.

MX Record

An MX record or *mail exchange record* is used to route email requests to mail servers. It has an additional priority property which allows multiple records for the same hostname to point to different servers and act as a backup. For example, a record with the priority of `10` will be used before a record with the priority of `20`.

See the section on MX records for more information.

Nameserver

The server which stores all DNS records for a particular domain. Think of this as a phonebook where DNS records are listed instead of people. You can only find the records you're looking for if you look in the right phonebook.

nslookup

Short for nameserver lookup, this is a tool used to ask questions and receive answers from DNS servers about their records. Included in the *BIND* suite of utilities, it is currently deprecated in favor of `dig` and `host`. While it can be used similarly to those tools, it also has an interactive mode:

```
$ nslookup
> set type=A
> donkeyrentals.com
```

See also: *BIND*, *dig*, *host*.

NS Record

A NS record or *nameserver record* points to the server that holds all other DNS records for a particular domain.

See also: *Nameserver*.

Public-key Infrastructure (PKI)

Public-key infrastructure. The system we (the internet) have agreed on to set up and maintain secure certificates. Kind of like how society decided we should fund the government, so we use the system of taxes. This system is used by browsers, servers, certificate authorities, and more to ensure we all have safe, secure browsing experiences.

See the section on public-key infrastructure for more information.

Private Key

A long, unique string of random looking numbers and letters that represents someone or something. Maybe that thing is a server, a website, or your computer. Imagine a physical lock and a key. In public key encryption, the *public key* is the lock, and the *private key* is the key. Confusing terms, yes, but that's more or less how it works. Never give your private key out or make it visible to the world.

See the section on public-key infrastructure for more information.

Public Key

As opposed to a *private key*, this is a different long, unique string of random-looking numbers and letters that represents someone or something. Feel free to give this out to anyone; it does not have to be private. Putting this somewhere else (like a server) usually gives you access, so long as you have the matching *private key*.

See the section on public-key infrastructure for more information.

Public-key Encryption

See *Public-key Cryptography*

Public-key Cryptography

The system for creating private and public keys. Imagine some crazy math that can take input like `admin@donkeyrentals.com` and spit out two files of garbage (public and private key). We use these keys as a form of security, which works well even if we don't understand the crazy algorithms behind encryption.

See also: *Public-key Infrastructure*

Quad-A Record

See *AAAA Record*

ping

A tool that makes a simple request to a server with an IPv4 address and returns how long that request took (among other things). It's a useful tool for checking to see if a server is up because, if the server is down, it will respond saying the ping timed out.

See the section on ping for more information.

ping6

Just like its sibling *ping*, but uses IPv6 addresses instead.

See also: *ping*

Registrar

This is a company whose job it is to keep records on behalf of customers. As a customer, you talk to the registrar to lease a domain.

Registry

A registry manages one or more top-level domains, like `com`, or `co.uk`, or `dentist`. They communicate with registrars to lease domains to customers. Registries themselves do not interact with the general domain-buying public.

Reverse DNS (RDNS)

Reverse DNS. This is when you try to get a domain name from an IP address. You can try this easily with dig:

```
$ dig +short -x 66.220.158.68

edge-star-mini-shv-07-frc3.facebook.com.
```

Root Domain

See *Apex Domain*.

Root Zone

The top level of the DNS hierarchy. Similar to how the `donkeyrentals.com` nameserver records are located at the `com` registry, `com`'s nameserver records are located at the registry called the root zone.

There are 13 root domains (`a.root-servers.net` through `m.root-servers.net`), which represent many servers (500+) around the globe.

SRV Record

An SRV record or *service record* is used to make a specific connection to an IP address using a specific port. Services might include a CalDAV, XMPP, or Minecraft server, among many others.

SRV records also offer two other attributes: priority and weight. When multiple requests are made to the service, they are sent to servers with lower-numbered priority first. When multiple servers have the same priority, requests are distributed across them according to their weight.

See the section on SRV records for more information.

SSL

Secure Sockets Layer. This protocol was replaced by *TLS* (Transport Layer Security) but is still commonly referred to when speaking about secure certificates. Thanks everyone for making it so confusing.

See the chapter on securing your website for more information.

Subdomain

A domain that is part of another domain. For example, `shop.donkeyrentals.com` is a subdomain of `donkeyrentals.com`, which is a subdomain of `com`. A subdomain is also technically a domain, and all domains (except for the root zone) are technically subdomains. Use subdomain when talking specifically about a domain that is defined by its parent domain:

> Welcome to donkeyrentals.com! Visit our shop at the subdomain shop.donkeyrentals.com.

TLD

Top-level Domain. These are the last part of a domain. These include generic TLDs (e.g., `com`, `org`, and `dentist`) and country code TLDs (e.g., `uk`, `dj`, and `io`).

See the IANA's Root Zone Database for a full list of TLDs.

TTL

Short for *time-to-live*, the amount of time left until a DNS record is re-fetched from its authoritative source, i.e., when the records cache will expire.

By the way, *live* rhymes with *give* not *dive*, as in "This is how long the data will live."

TXT Record

Text records are for storing arbitrary data with a hostname. Example:

- **Hostname**: `message`
- **IP Address**: `Hello`

This may seem like a trivial example, and it is. However, real-world use cases do exist, such as verifying an email server or providing proof of ownership for a TLS record.

See the section on TXT records for more information.

WHOIS

A protocol and tool that retrieves information about who is in control of a domain name. Confusingly, the protocol (WHOIS) and the tool (`whois`) are different; `whois` uses the WHOIS protocol to get domain information.

Basic usage:

```
$ whois donkeyrentals.com
```

See the section on whois for more information.

X.509

X.509 is a list of standards that specify public key certificates, certificate revocation lists, attribute certificates, and a certification path validation algorithm. It's like being in a secret club, but, instead of building a treehouse, you agree to use certain methods to create and structure certificates and how to communicate with certificate authorities. TLS/SSL uses X.509 certificates.

This is in contrast to a model such as PGP, where you get together with all your nerdy friends and sign each other's certificates. X.509 certificates rely on being signed by global certificate authorities to be valid, among other formatting requirements.

See the section on securing your website for more information.

Appendix

Installing tools

We will be installing the software used in this book with a package manager. Unix and Linux operating systems often come with a package manager installed. Package managers are handy because, in addition to installing software, they install supporting packages, keep an up-to-date list of safe software packages, and make it easy to uninstall software later.

Think of a package like an application: software written to do a specific set of tasks. The difference is that these packages are almost always used from the command prompt instead of opening up a graphical interface. Installing graphical applications via a package manager is not unheard of, however.

The tools used in this book (`dig`, `host`, `openssl`, `nslookup`, and `whois`) are all available via a package manager. While this is incredibly convenient, it is even *more* convenient that many of these tools are already installed.

From your shell or command prompt, you can use the `which` command to see if a tool is installed:

```
$ which dig

/usr/bin/dig
```

Or the `where` command on Windows:

```
$ where dig

C:\ProgramData\chocolatey\bin\dig.exe
```

You can replace `dig` with any of the following:

- `host`
- `openssl`
- `nslookup`
- `whois`

If the tool is installed, it will return its location on the system. If not, it will say `<command> not found`. If that is the case, you can use your package manager to install it, as described below.

Mac and Windows

Neither Mac OS X nor Windows comes with a package manager. Luckily, this is pretty easy to resolve. If you are using either of these systems, check out the *Package Management for Mac OS X* or *Package Management for Windows* section before moving on to the next section.

Installation

Below are the basic commands to install each tool on seven different operating systems. For the purposes of this book, and perhaps the world at large, this is all you will need.

BIND (which includes dig, host, and nslookup)

- Arch Linux: `pacman -S bind`
- CentOS: `yum install bind`
- Debian: `apt-get install bind9`
- Gentoo: `emerge -atv net-dns/bind`

- Fedora: `yum install bind`
- Mac OS X: `brew install bind`
- Ubuntu: `apt-get install bind9`
- Windows: `choco install bind-toolsonly`

whois

- Arch Linux: `pacman -S whois`
- CentOS: `yum install whois`
- Debian: `apt-get install whois`
- Gentoo: `emerge -atv net-misc/whois`
- Fedora: `yum install whois`
- Mac OS X: `brew install whois`
- Ubuntu: `apt-get install whois`
- Windows: `choco install whois`

OpenSSL

- Arch Linux: `pacman -S openssl`
- CentOS: `yum install openssl`
- Debian: `apt-get install openssl`
- Gentoo: `emerge -atv dev-libs/openssl`
- Fedora: `yum install openssl`
- Mac OS X: `brew install openssl`
- Ubuntu: `apt-get install openssl`
- Windows: `choco install openssl.light`

Note for OpenSSL on Windows: You will also need to add OpenSSL's `bin` *directory to your path. Copy and paste the following in the Command Prompt:*

```
$ setx path "%PATH%;C:\Program Files\OpenSSL\bin"
```

Installation check

To ensure the tools installed properly, run a quick `which <command>` or `where <command>`. For a slightly more involved check, invoke the command in its simplest form:

- dig: `dig -v` – Displays the current version of the dig command.
- host: `host` – Displays list of command options.
- openssl: `openssl version` – Displays the current version of the `openssl` command.
- nslookup: `nslookup` – Enters interactive mode; type `exit` to close.
- whois: `whois` – Displays list of command options.

Package Management for Mac OS X

Mac OS X doesn't come with a package manager installed. I highly recommend homebrew, which can be installed with the following command:

```
$ /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/i
```

Once it completes, you can now use the `brew` command to install, uninstall, update, and perform many other functions on tons of packages for your system:

```
$ brew install whois
$ brew uninstall whois
$ brew search openssl
$ brew upgrade openssl
```

That's it! Now you should be all set to install the tools.

Package Management for Windows

The bad news? Windows does not come with a package manager. The good news? It is pretty easy to install one. In particular, we are going to install chocolatey.

Open your Command Prompt by using the Start menu to search for `Command`. When you see the Command Prompt, right click it and "Run as administrator."

Now, copy and paste the following crazy looking command into it (trust me, don't even try to type it out):

```
@powershell -NoProfile -ExecutionPolicy Bypass -Command "iex ((new-object net.webclient).Dow
```

This will take some time, perhaps a minute or two for a modern-ish computer. If it takes significantly longer, try restarting, and make sure you open the Command Prompt as an administrator.

Once it is done, you will have the chocolatey package manager installed on your system. Use the `choco` tool to issue basic commands. For example:

```
$ choco install whois
$ choco uninstall whois
$ choco search openssl
$ choco upgrade chocolatey
```

I also recommend enabling the `allowGlobalConfirmation` feature for chocolatey. If you do not, it will ask if you want to install each package every time:

```
$ choco feature enable -n=allowGlobalConfirmation
```

To disable this feature, run:

```
$ choco feature disable -n=allowGlobalConfirmation
```

Now you can install tons of great packages!

Additional installation

This book makes heavy use of the BIND suite of tools, like `dig`, `host`, and `nslookup`. Chocolatey can install them, but they will also need some additional supporting software. Specifically, the succinctly named *Visual C++ Redistributable for Visual Studio 2012 Update 4*.

Installation is pretty simple:

1. Visit this website.
2. Download the x64 or x86 version of the software based on your system architecture. If you are not sure what system architecture you have, follow these steps:
    a. On Windows 7, open the Start menu and right-click on `Computer`, then click `Properties`
    b. On Windows 10, right click the Start menu and click `System`
    c. In the System section, `System type` should say **32-** or **64-bit Operating System**.
    d. Download x64 for 64-bit systems and x86 for 32.
3. Double click the installer and let it do it's thing.
4. `dig`, `host`, and `nslookup` should work without issue.

Recommended Registrars

Feeling overwhelmed by the number of registrars out there? Here are three that I recommend:

- Hover
- DNSimple
- Gandi

Hover has a great control panel and awesome customer support if you need it. The only thing they lack is ALIAS/ANAME record support. DNSimple lives up to its name and supports many features, including ALIAS/ANAME. They cost slightly more than other registrars, but it's worth it if you're a bit intimidated by managing a domain name. Gandi comes very highly recommended from colleagues.

Conclusion

Afterword

So what's left? You read this book, learned some things about DNS, TLS certificates, and domains in general. Congratulations! I also have a bit of advice for you.

I've spent a lot of time reading through Server Fault to understand the particulars of a topic. It's a great Q&A site where you can often find your question has already been asked and answered. If not, you can sign up and ask for free.

I find that I learn best when I'm experimenting. If you have the means, I highly suggest getting a spare domain to play around with. Set up a quick site and just go crazy with different record types. Query them with `dig` and see what you can get out of them. Redirect to other sites with CNAMEs, chat with friends via TXT records, etc. This can help solidify your understanding of the whole domain process.

If you really want to dive into the deep technical details of how DNS or TLS works, RFCs (requests for comments) are the way to go. An RFC document is usually made by a standards committee to decide how certain technology will work. It includes everything a developer would need to create software that works with other like-minded software. RFCs are playbooks that keep distributed systems like DNS working, even when multiple different, often competing, organizations write software for a purpose.

Wikipedia keeps a great list of DNS related RFCs here.

Last, and most importantly, continue to be curious. Curiosity wrote this book, curiosity likely got you through this book, and curiosity will keep you learning, inspecting, and researching to find out more. That is your most powerful tool. Use

it to its fullest extent.

Thanks

First of all, thank *you* dear reader. I wrote this book so you could absorb and enjoy it. If you've gotten this far, you probably don't hate it, which is a compromise I can settle for.

But also, I must thank a number of great people for helping me with this book:

- Caleb Thompson and thoughtbot, for thinking, "Yeah, you're probably qualified to write a book." Caleb especially, for answering all my inane questions and encouraging me to write.
- Mike Burns, for tearing apart my god-awful first draft of the "Securing Your Website" chapter.
- Frank Wang, for patiently explaining how public-key infrastructure works and reading through the "Securing Your Website" chapter.
- My wife, Elizabeth, for reading a chapter once and hating it, but loving me despite that.
- My mother, Kathie, who read early drafts and even opened the command line a few times to test things.
- Gabe Berke-Williams, for laughing at my incredibly dumb jokes and testing absurd scenarios with me.
- Josh, Diana, and Tute, for taking time to read through chapters and fixing my grammar and spelling mistakes.