Networking

Certbot SSL Certificates

Its important to encrypt your web traffic to keep you and anyone who passes information through your website secure.

To install Certbot and generate an SSL certificate, run the below commands.

```
curl -o- https://raw.githubusercontent.com/vinyll/certbot-install/master/install.sh | bash
# Generate a certificate, but don't do any automatic NGINX configuration
sudo certbot certonly --nginx -d domain.com -d www.domain.com
```

Since we passed the `certonly` argument to `certbot`, there will not be any automatic configuration of our NGINX server to use SSL. In my experience using this automatic configuration tool on an NGINX server that has already been modified from the default settings doesn't work very well, so I'll explain the required changes to `nginx.conf` later on this page.

There are a few benefits to using Certbot. Your certificates will automatically be renewed when nearing expiration, and it can configure several different web servers to use the new SSL certificate automatically. I personally do the configuring manually, but on a new server the automatic configuration might be a useful feature to you!

To check on the time left until certbot renews -

```
sudo systemctl status certbot.timer
```

Dry run renew with your current configuration -

```
sudo certbot renew --dry-run
```

Check installed certificates on this system -

```
sudo certbot certificates
Saving debug log to /var/log/letsencrypt/letsencrypt.log
OCSP check failed for /etc/letsencrypt/live/domain.com/cert.pem (are we offline?)


- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Found the following certs:
  Certificate Name: domain.com
    Domains: domain.com www.domain.com
    Expiry Date: 2020-08-18 03:00:10+00:00 (INVALID: EXPIRED)
    Certificate Path: /etc/letsencrypt/live/domain.com/fullchain.pem
    Private Key Path: /etc/letsencrypt/live/domain.com/privkey.pem
  Certificate Name: other-domain.com
    Domains: other-domain.com www.other-domain.com
    Expiry Date: 2020-08-24 22:16:30+00:00 (VALID: 6 days)
    Certificate Path: /etc/letsencrypt/live/other-domain.com/fullchain.pem
    Private Key Path: /etc/letsencrypt/live/other-domain.com/privkey.pem
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

NGINX SSL Setup

Now we need a webserver to redirect traffic over https. The below nginx configuration is verified to be working on Ubuntu 20.04 using certbot certificates to decrypt the traffic on default port 80, then passing it to a container hosted locally on a specific port. See the NGINX Book for more details on configuring nginx.

```
user www-data;
worker_processes auto;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;

events { }

http {
  include mime.types;

  # Redirect root domains
  server {
    listen 80;
    server_name domain.com www.domain.com;
    return 301 https://www.domain.com$request_uri;
  }


  # SSL - domain.com
  server {
    server_name domain.com www.domain.com;
    server_tokens off;
    listen 443 ssl;
    ssl_certificate /etc/letsencrypt/live/domain.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/domain.com/privkey.pem;

    # Pass to container
    location / {
      include proxy_params;
      proxy_pass http://localhost:1234/;
    }
  }

}
```

Errors

Sometimes, if your webserver is running already, you may see the following error

```
Attempting to renew cert (domain.com) from /etc/letsencrypt/renewal/domain.com.conf produced an unexpected error: Problem binding to port 80: Could not bind to IPv4 or IPv6.. Skipping.
```

To fix this, we can use certbot's `--pre-hook` and `--post-hook` options

```
sudo certbot renew --dry-run --pre-hook 'service nginx stop' --post-hook 'service nginx start'
```

You can also add executables to the `/etc/letsencrypt/renewal-hooks/pre` and `/etc/letsencrypt/renewal-hooks/post` directories and certbot will automatically handle executing them on a request to renew certificates, removing the need to specify these arguments each time. See Official Certbot Documentation for more info. An important point made there can be seen in the quote below

> These hooks are run in alphabetical order and are not run for other subcommands. (The order the hooks are run is determined by the byte value of the characters in their filenames and is not dependent on your locale.)
>
> Hooks specified in the command line, configuration file, or renewal configuration files are run as usual after running all hooks in these directories.

More help can be found within a termianl with `sudo certbot --help renew`

Proxy Servers

Proxies

There are two primary types of proxies, forward and reverse proxy. A proxy masks the IP address of a client or server by routing requests through another server. Proximity to this remote server and the average load on this server can impact network speeds.

Forward Proxy

A forward proxy is often used to mask a client's IP address by routing requests through another server. The purpose of a forward proxy is to evade detection or firewall rules that would otherwise impact the client's ability to visit the service. There are Web, HTTP, and SOCKS proxies. Avoid free proxy services as they are a popular form of data collection used by malicious parties to collect and potentially exploit your internet traffic. Forward proxies are not only used for malicious purposes, and can also be used to better organize and enforce server infrastructure rules. For example, routing requests from a cluster of internal servers through a forward proxy enables you to configure the single forward proxy instead of each individual server. This can be used to funnel internal traffic to the external web and better control the traffic and behavior of your infrastructure.

Reverse Proxy

A reverse proxy is a common tactic employed by organizations to implement load balancing features on high-traffic web services. A service which generates a lot of traffic can route requests from clients and forward them to any one of a large number of proxy servers. By diverting requests to through the proxy to the server with the smallest load, we reduce the average load on the server and can maintain a more consistent service with speed and availability. There is also the added security benefit of hiding your server's IP address and protecting it from malicious scans and activity by using a reverse proxy, since the proxy server can be used to filter out bad requests and protect your infrastructure. NGINX, Apache, and HAProxy are all popular forms of reverse proxy servers.

Resources and Links

Linuxbabe - Forward & reverse Proxies

JSCAPE - Forward & reverse Proxies

Web Servers

NGINX

Install

Install nginx by running the commands below

```
sudo apt update && sudo apt upgrade
sudo apt install nginx
```

Configure SSL

SSL certificates are limited, see the LetsEncrypt documentation on SSL rate limits for more information. Take notice of the section about renewals - to avoid regenerating certificates during testing, run `sudo certbot certonly --dry-run -d domain.com -d www.domain.com`

Before we pass any traffic, we should configure SSL for any domains we want to serve on this host. To use LetsEncrypt and Certbot to do this, run the commands below.

Certbot installation instructions have changed

The new method of installation, as explained in the above link, is using `snap`. I very much dislike snap, because I've been on systems with limited resources and have experienced snap causing poor performance, especially when installing larger applications with it.

In any case, the new method of installation is below. `certbot` is an apt package, but the official instructions do not recommend to use that apt package (so why is it there o.o)

If you installed with apt, remove certbot first, then reinstall with snap.

```
sudo apt remove certbot
sudo snap install --classic certbot
sudo certbot certonly --nginx
```

Reading the output generated, we can see where our certificates were created. **Take note of these paths**, you will need to refer to these certificates within your `/etc/nginx/nginx.conf`

```
Successfully received certificate.
Certificate is saved at: /etc/letsencrypt/live/git.shaunreed.com/fullchain.pem
Key is saved at:         /etc/letsencrypt/live/git.shaunreed.com/privkey.pem
This certificate expires on 2022-04-25.
These files will be updated when the certificate renews.
Certbot has set up a scheduled task to automatically renew this certificate in the background.
```

There are a few benefits to using Certbot. Your certificates will automatically be renewed when nearing expiration, and it even configures nginx for you automatically.

Instead of using the default configuration Certbot creates, you can make one yourself. Below, we create our own nginx configuration from scratch which still uses Certbot to manage SSL certificates.

Basic NGINX Settings

A virtual host in NGINX serves content based on settings found within `/etc/nginx/nginx.conf` , we can use these settings to do things like handle SSL and pass traffic to other hosts if using a specific sub domain.

These settings can be modified to suit the needs of a basic host serving one page or application.Below, we route traffic to a docker container running on a localhost port.

```
user www-data;
worker_processes auto;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;

events { }

http {
  include mime.types;

  # Redirect root domains
  server {
    listen 80;
    server_name domain.com www.domain.com;
    return 301 https://www.domain.com$request_uri;

  }



  # SSL - domain.com
  server {
    server_name domain.com www.domain.com;
    server_tokens off;
    listen 443 ssl;
    ssl_certificate /etc/letsencrypt/live/domain.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/domain.com/privkey.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    # Pass to container
    location / {
      include proxy_params;
      proxy_pass http://localhost:1234/;
    }

  }

}
```

Multiple Domains

If serving multiple domains over SSL on one host, see the configuration below for a basic example. It should look fairly similar to the above.

```
user www-data;
worker_processes auto;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;

events { }
```

```
http {
  include mime.types;

  # Redirect root domains
  server {
    listen 80;
    server_name domain.com www.domain.com;
    return 301 https://www.domain.com$request_uri;
  }

  server {
    listen 80;
    server_name domain2.com www.domain2.com;
    return 301 https://www.domain2.com$request_uri;
  }

  # SSL - domain
  server {
    server_name domain.com www.domain.com;
    server_tokens off;
    listen 443 ssl;
    ssl_certificate /etc/letsencrypt/live/domain.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/domain.com/privkey.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    # Pass to container
    location / {
      include proxy_params;
      proxy_pass http://localhost:1234/;
    }

  }

  # SSL - domain2
  server {
    server_name domain2.com www.domain2.com;
    server_tokens off;
    listen 443 ssl;
    ssl_certificate /etc/letsencrypt/live/domain2.com/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/domain2.com/privkey.pem; # managed by Certbot
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    # Pass to second container
    location / {
      include proxy_params;
      proxy_pass http://localhost:4321/;
    }

  }
}
```

Above, we serve two different applications running on different ports depending on the url request.

Custom logging

Sometimes, especially when hosting multiple domains on one box, you may want to separate the NGINX logs based on return code *and* host / domain name referenced. Below we see a `nginx.conf` which enables this feature -

```
user www-data;
worker_processes auto;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;

events { }

http {
  include mime.types;

  # Redirect root domains for first server
  server {
    listen 80;
    server_name firstsite.com www.firstsite.com;
    return 301 https://www.firstsite.com$request_uri;
  }

  # Redirect root domains for second server
  server {
```

```
  listen 80;
  server_name secondsite.com;
  return 301 https://secondsite.com$request_uri;
}

# Map the 100-200 error codes to $oks
map $status $oks {
  ~^[1-2] 1;
  default 0;
}

# Map the 400-500 error codes to $errs
map $status $errs {
  ~^[4-5] 1;
  default 0;
}

# Map the 300 error codes to $redir
map $status $redir {
  ~^[3] 1;
  default 0;
}

# SSL - firstsite
server {
  server_name firstsite.com www.firstsite.com;
  server_tokens off;
  listen 443 ssl;
  ssl_certificate /etc/letsencrypt/live/firstsite.com/fullchain.pem;
  ssl_certificate_key /etc/letsencrypt/live/firstsite.com/privkey.pem;

  # Configure Server-wide logging
  access_log /var/log/nginx/access.log;
  error_log /var/log/nginx/error.log;

  # Configure logs for this domain
  access_log /var/log/nginx/firstsite.log;

  # Configure return-specific logging
  access_log /var/log/nginx/firstsite.access combined if=$oks;
```

```
    access_log /var/log/nginx/firstsite.error combined if=$errs;
    access_log /var/log/nginx/firstsite.redir combined if=$redir;

    # Pass to firstsite container
    location / {
      include proxy_params;
      proxy_pass http://localhost:4321/;
    }
   }



  # SSL - secondsite
  server {
    server_name secondsite.com;
    server_tokens off;
    listen 443 ssl;
    ssl_certificate /etc/letsencrypt/live/secondsite.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/secondsite.com/privkey.pem;

    # Configure Server-wide logging (to create one log to monitor with fail2ban, ossec, etc)
    access_log /var/log/nginx/access.log;
    error_log /var/log/nginx/error.log;

    # Configure logs for this domain
    access_log /var/log/nginx/secondsite.log;

    # Configure return specific logginng if they match the statuses we mapped
    access_log /var/log/nginx/secondsite.access combined if=$oks;
    access_log /var/log/nginx/secondsite.error combined if=$errs;
    access_log /var/log/nginx/secondsite.redir combined if=$redir;

    # Pass to second container
    location / {
      include proxy_params;
      proxy_pass http://localhost:1234/;
    }

  }
}
```

Now after running `systemctl reload nginx` you should notice there are logs for each domain being created according to your organization above. It can be useful for smaller setups, but would quickly get out of hand with large amounts of traffic, I'd imagine.

No SSL

This is at the bottom of the page for a reason. This should only be used for testing, but you can get away with a very simple nginx configuration if you don't use SSL encryption -

```
user www-data;
worker_processes auto;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;

events { }

http {
  include mime.types;

  # Redirect root domains
  server {
    listen 80;
    server_name domain.com www.domain.com;

    # Pass to container port
    location / {
      include proxy_params;
      proxy_pass http://localhost:1234/;
    }

  }
}
```

Apache

Changing your Apache Server Root Directory

Your Apache installation determines what content to serve based on the DocumentRoot Default directory.

```
DocumentRoot Default = /var/www/html


-document root configured in the following files:
sudo vim /etc/apache2/sites-available/000-default.conf
sudo vim /etc/apache2/apache2.conf
```

Application Port Settings

After installing Apache, there will be a default configuration file located in `/etc/apache/enabled-sites/` that contains some VirtualHost configuration settings. It can be modified using Apache modules to allow Apache to serve our application as our root domain. To do this, we will need to enable one of the following Apache mods using `sudo a2enmod <module name>`

So, we run..

```
sudo a2enmod proxy_http
sudo a2enmod mod_proxy
```

After these two modules install, set up your VirtualHost configuration within the /etc/apache/enabled-sites/ directory as follows:

```
<VirtualHost *:80>
    ServerAdmin me@mydomain.com
    ServerAlias mydomain.com
    ServerName www.mydomain.com
    ProxyPreserveHost On


    # setup the proxy
    <Proxy *>
        Order allow,deny
        Allow from all
    </Proxy>

    ProxyPass / http://www.mydomain.com:8888/

    ProxyPassReverse / http://www.mydomain.com:8888/

</VirtualHost>
```

To apply these settings, run `sudo systemctl restart apache2.service` to restart Apache and reload your new configuration. If this command returns an error, look closer at your config settings and be sure you enabled the necessary modules.

Protocols

Subnetting

Subnetting is the organization of hosts within smaller networks using subnet masks, which allows us to route packets to their destination more efficiently. This approach is similiar to what is employed by delivery services, where the package is first routed to a sorting destination within a certain range of the final destination. By routing packets through the nearest subnet we reduce the overhead of traveling long distances and instead we make smaller hops between organized subnets.

For a single IP address there are 32 bits. This means that the IP `0.0.0.0` is equal to `00000000.00000000.00000000.00000000` in bits. This is an important concept to keep in mind, as it will help to understand subnet masks later on. When converting these to decimal IP addresses, we should keep in mind that each octet (set of 8 bits) starts at 2^7 and ends at 2^0. So, the IP `11000000.00110000.00001111.00111100` is calculated as follows, where each step is a single octet converted to decimal form.

1. `11000000` = $1x2^7 + 1x2^6 + 0x2^5 + 0x2^4 + 0x2^3 + 0x2^2 + 0x2^1 + 0x2^0 = 192$
2. `00110000` = $0x2^7 + 0x2^6 + 1x2^5 + 1x2^4 + 0x2^3 + 0x2^2 + 0x2^1 + 0x2^0 = 48$
3. `00001111` = $0x2^7 + 0x2^6 + 0x2^5 + 0x2^4 + 1x2^3 + 1x2^2 + 1x2^1 + 1x2^0 = 15$
4. `00111100` = $0x2^7 + 0x2^6 + 1x2^5 + 1x2^4 + 1x2^3 + 1x2^2 + 0x2^1 + 0x2^0 = 60$

So the IP `11000000.00110000.00001111.00111100` represented in bits is equal to 192.48.15.60, which is a class C IP address.

There are 5 classes of IP addresses, which are outlined below.

Class A - From left-to-right, everything before the first `.` identifies the network, and all other sections of the IP represent different devices on that network. For example, `127.56.98.102` is on the `127` network, and the device is `56.98.102`. In bits, a class A IP must begin with a single `0`, leaving 7 more bits available to identify the network, and the remaining 24 bits to identify devices within the network. So, with a class A IP we have 2^7 (128) possible network ranges, where each range has 2^{24} (16,777,216) IPs available to assign to devices. Thus, class A IPs start at `0.0.0.0` and end at `127.255.255.255`. Class A IPs are used for very large networks, notably those deployed by Internet Service Providers (ISPs). This is where your public IP adddress lives.

Class B - From left-to-right, everything before the second `.` identifies the network, and all other sections of the IP represent different devices on that network. For example, `128.123.58.19` is on the `128.123` network, and the device is `58.19`. In bits, a class B IP must begin with `10`, leaving 14 more bits available to identify the network, and the remaining 16 bits to identify devices within the network. So, with a class B IP we have 2^{14} (16,384) possible network ranges, where each range has 2^{16} (65,636) IPs available to assign to devices. Thus, class B IPs start at `128.0.0.0` and end at

`191.255.255.255`. These IPs are often used for large networks deployed by enterprises or organizations with large infrastructure. This is where a company like Google or Amazon would organize their infrastructure.

Class C - From left-to-right, everything before the third `.` identifies the network, and all other sections of the IP represent different devices on that network. For example, `192.48.15.60` is on the `192.48.15` network, and the device is `60`. In bits, a class C IP must begin with `110`, leaving 21 more bits available to identify the network, and the remaining 8 bits to identify devices within the network. So, with a class C IP we have 2^{21} (2,097,152) possible network ranges, where each range has 2^8 (256) IPs available to assign to devices. Thus, class C IPs start at `192.0.0.0` and end at `223.255.255.255`. These IPs are often used for small business and home networks, and is where your local IP address lives.

Class D - Class D IPs use all 32 bits for network addressing but they must begin with `1110`, leaving 28 more bits available to identify the multicast IP. For example, `227.63.12.126` is just the `227.63.12.126` multicast IP address with no further identification for a host device. Thus, class D IPs start at `224.0.0.0` and end at `239.255.255.255`. These addresses are used for multicasting operations and there are no host devices within this IP class. So, with a class D IP we have 2^{28} (268,435,456) possible multicast IPs.

Class E - Class E IPs use all 32 bits for network addressing but they must begin with `1111`, leaving 28 more bits available to identify the IP. So, with a class E IP we have 2^{28} (268,435,456) possible IPs. Thus, class E IPs start at `240.0.0.0` and end at `255.255.255.255`. But these addresses are not used at all and considered invalid, thus there are no host devices or networks within this IP class. The only exception is the broadcast IP address which is the same on every network - `255.255.255.255`

IP prefixes represent a count of bits used to identify a network, which helps to define the subnet of available hosts. For example, `x.x.x.x/4` uses 4 leading bits to identify the network which can host $2^{32 - 4}$ IP addresses (268,435,456)

All IP subnets possible can be seen in this table, provided at Freecodecamp - Subnet Cheatsheet

CIDR	SUBNET MASK	WILDCARD MASK	# OF IP ADDRESSES	# OF USABLE IP ADDRESSES
/32	255.255.255.255	0.0.0.0	1	1
/31	255.255.255.254	0.0.0.1	2	2*
/30	255.255.255.252	0.0.0.3	4	2
/29	255.255.255.248	0.0.0.7	8	6
/28	255.255.255.240	0.0.0.15	16	14
/27	255.255.255.224	0.0.0.31	32	30
/26	255.255.255.192	0.0.0.63	64	62
/25	255.255.255.128	0.0.0.127	128	126
/24	255.255.255.0	0.0.0.255	256	254
/23	255.255.254.0	0.0.1.255	512	510
/22	255.255.252.0	0.0.3.255	1,024	1,022
/21	255.255.248.0	0.0.7.255	2,048	2,046
/20	255.255.240.0	0.0.15.255	4,096	4,094
/19	255.255.224.0	0.0.31.255	8,192	8,190
/18	255.255.192.0	0.0.63.255	16,384	16,382
/17	255.255.128.0	0.0.127.255	32,768	32,766
/16	255.255.0.0	0.0.255.255	65,536	65,534
/15	255.254.0.0	0.1.255.255	131,072	131,070
/14	255.252.0.0	0.3.255.255	262,144	262,142
/13	255.248.0.0	0.7.255.255	524,288	524,286
/12	255.240.0.0	0.15.255.255	1,048,576	1,048,574

PCMag - IP Addresses

paessler - IP Addresses

Cloudflare - Subnetting

Pearson - Subnetting

DNS

Basic Settings

To configure basic DNS for a new domain, we will only really need to create two records, after ensuring our nameservers are pointed to the correct location.

If you are just messing around with NGINX or Apache, there is no real need to purchase a domain to simply resolve your IP with DNS. Check out Freenom for a free domain name, granted it may not be your first choice - but they provide a wide range of **free domains** for up to 12 months.

Nameservers

A nameserver defines the path the DNS will take to resolve your domain name's IP address. If you purchased your domain already, chances are you created an account with the vendor you purchased from. Login to this account, and locate a 'DNS Records / Settings' panel to modify your DNS records using the vendors supplied control panel. If you would rather use another control panel, for example DigitalOcean, you would need to login to your domain provider's control panel and alter your domain's nameservers to reflect the below -

```
ns1.digitalocean.com
ns2.digitalocean.com
ns3.digitalocean.com
```

This allows your domain to resolve using the appropriate servers on which we have set our DNS records using their respective control panels.

DNS Records

To get started using our new domain, we will only need the below basic DNS settings -

Type	Hostname	Value	TTL
A	www.website.com	0.0.0.0	3600
A	@.website.com	0.0.0.0	3600
A	*.website.com	0.0.0.0	3600

These basic settings will allow you to further configure DNS on the host directly using a webserver if you so choose, or if you'd rather the interface using the control panel associated with your nameservers is fine as well.

DNS Record Types / Definitions

A Records

An A record maps an IPv4 address to a domain name. This determines where to direct any requests for a domain name.

AAAA Records

An AAAA record, also called a Quad A record, maps an IPv6 address to a domain name. This determines where to direct requests for a domain name in the same way that an A record does for IPv4 addresses.

CNAME Records

A CNAME record defines an alias for an A record; it points one domain to another domain instead of to an IP address. When the associated A record’s IP address changes, the CNAME will follow to the new address.

MX Records

An MX record specifies the mail servers responsible for accepting email on behalf of your domain. Providers often make multiple name servers available so that if one is offline, another can respond. Each server needs its own MX record.

NS Records

An NS record specifies the *name servers*, or servers that provide DNS services, for a domain or subdomain. You can use these to direct part of your traffic to another DNS service or to delegate DNS administration for a subdomain.

Web Query Path

In general, when we visit google.com for example, these are the steps that are taken to resolve the destination IP address, which is resolved from right-to-left.

1. Client queries google.com via some application
2. The browser checks if the DNS entry is within the local cache, then the OS checks it's local cache. If either is found to exist it returns it to the client immediately and the DNS resolution is complete; Otherwise, continue to next step
3. Client query is sent to router on the LAN
4. The router sends the query to the DNS Recursive Resolver if there is no entry in the cache

5. The DNS Recursive Resolver routes the request the the DNS Root Nameserver (`.`)
6. The Root server responds with the Top Level Domain (TLD) DNS server (`.com`, `.net`, `.io`, etc)
7. The DNS Recursive Resolver routes the query to the TLD server
8. The TLD server responds with the destination IP of the domain's nameserver (`shaunreed.com`, `google.com`, etc)
9. The Recursive Resolver sends the request to the domain's Authoratative Nameserver
10. If the request has a subdomain (like `git.shaunreed.com`), the Authoratative Nameserver returns the final DNS Authoratative nameserver that is responsible for storing the subdomain's CNAME record. If there is no subdomain, this step is skipped and the destination IP is returned
11. The final DNS Authoratative Nameserver returns the destination IP address back to the Recursive Resolver
12. The Recursive Resolver returns the resolved destination IP address to the router which initially sent the query
13. The router delivers the resolved destination IP to the client IP that initially requested it on the router's LAN
14. The client caches the result in the OS and web browser, assigning a Time To Live (TTL) value to indicate when the cached result should expire

Once all of these steps are completed, or a cached result is returned, the client's actual request is carried out to the destination. This is true whether the request is HTTP, HTTPS, FTP, or any other protocol within the Application layer of the TCP/IP or OSI network models.

On Ubuntu Linux systems, Mozilla stores the cache in `~/.cache/mozilla`, where everything from DNS resolutions, mozilla settings, and images / thumbnails are cached.

On Ubuntu Linux systems, by default the systemd-resolved service is enabled but the /etc/systemd/resolved.conf has a default `Cache` value of `no-negative`, which means the service will not cache DNS resolutions on an OS level.

Resources and Links

Cloudflare - DNS

TCP/IP

Transmission Control Protocol / Internet Protocol describes the standardization of packet construction and routing between software and network destinations, respectively. This means that TCP/IP allows software to be created without the overhead of determining how communication packets should be constructed for each individual piece of software. This abstraction simplifies the development process by providing a well-defined process for all software that intends to communicate between devices within the network. This simplification is also seen on the receiving end, since we know how the packets were constructed before being broken down into bits and sent over the wire, we can also reconstruct them in a similar manner on the receiving end of the transmission.

The term TCP/IP describes the connectivity between the TCP and IP protocols. The TCP protocol describes how packets should be constructed and maintains the reliable connection for the data to be sent. The IP protocol describes where to deliver the data, and handles routing to the appropriate destination. When used together, we have a reliable connection between two machines across the network.

The joining of the two individual protocols gives us TCP/IP which contains the following layers, in this order. Note that this order represents the sending device, and receiving devices will actually traverse these layers backwards while reconstructing the data to the same format in which it was originally sent.

4. Application
3. Transport
2. Internet
1. Host-to-Network

These two protocols are used together so often that the term TCP/IP is commonly used in place of referring to TCP or IP individually.

Application

Describes the application or interface that the user interacts with, such as a web browser, chat service, or email application.

This layer depends upon the subsequent layers to handle packet construction, routing, and delivery via physical infrastructure.

Protocols in this layer are HTTP, HTTPS, DNS, FTP, and Telnet

Transport

The Transport layer is responsible for providing a reliable connection between two devices on a network. With a reliable connection the data is broken down into smaller packets which can then be sent over the wire and reconstructed on the receiving end.

Upon receiving the packets, the recipient sends an ACK (acknolwedgement) of the packet having been received, and then the sender replies with an additional ACK to indicate that it is aware the packet reached it's destination.

This layer depends on subsequent layers to route the packet to the correct destination, and to provide the physical infrastructure which connects the devices within the same network.

Protocols for this layer are TCP and UDP, or Transmission Control Protocol and User Datagram Protocol.

Internet

The Internet layer is sometimes referred to as the Network layer due to it being renamed within the OSI model, which we will talk about later on this page. This layer is responsible for routing the packets to the correct destination, and handles efficiently delivering the packets by finding the shortest or least-used path available between the two devices on the network.

This layer depends on the Datalink layer to provide the physical infrastructure that allows connectivity between the two devices on the network.

The IP protocol is used in this layer, along with ARP and ICMP

Host-to-Network

The Host-to-Networ layer is sometimes referred to as the Data Link layer, Network Access layer, or the Physical layer. This layer is responsible for facillitating the physical connections and infrastructure between two devices within the same network. Examples of this

Protocols for this layer are DSL, LAN, SATNET, SONET, WiFi (802.11), and Ethernet (802.3)

Resources and Links

TechTarget - TCP/IP

Avast - TCP/IP

Cloudflare - TCP/IP

OSI Model

OSI

Open System Interconnection Model breaks the TCP/IP protocol down into more specific layers with more specific responsibilities.

The layers of the OSI model appear in this order when information is being sent out from the host machine. If information is being received, the layers would be traversed in reverse order from 1-7

7. Application
6. Presentation
5. Session
4. Transport
3. Network
2. Data Link
1. Physical

Application

The Application layer is the interface or software that the user interacts with in order to send communications over the network. This could be a web browsers, chat services, or email clients. This layer is where the application's protocol lives.

This layer is generally the same as the Application layer of the TCP/IP model.

Protocols in this layer are HTTP, HTTPS, DNS, SMTP, ICMP, FTP, and Telnet

Presentation

The Presentation layer is responsible for converting data into a standardized format which can optionally be encrypted or compressed before being sent across the wire. Thus, the Presentation layer is responsible for converting data into a format which can either be used by the Application layer, or passed down to the Session layer to later be sent over the wire to another device on the network.

If the application wanted to encrypt the data, this layer would do so. Similarly, if the data this layer receives is encrypted, this layer decrypts the data so the receiving Application can use it.

Session

The Session layer is responsible for establishing and maintaining a reliable connection between *applications and/or devices* on the network. This layer is not always used, but services such as streaming video and audio would use this layer heavily, or in the case of monitoring remote system resources or logs.

If you are familiar with sockets in programming, think of this layer as the creation of a socket between a local or remote application that may either be on the machine locally or on a remote server.

This layer is also responsible for creating checkpoints during large data transfers, which enables the download to pick up where it left off in the case of an interruption. If it were not for this layer, the download would need to be restarted from the beginning in order to ensure that all the information was sent over the wire.

Transport

The Transport layer is responsible for establishing and maintaining a reliable end-to-end connection between *two devices on different networks*. If the devices are on the same network, this layer is not used.

When data is being sent, this layer breaks the data down into smaller segments before sending it across the wire. When data is being received, this layer reconstructs the data into its original format.

By the sender, multiplexing is used to package local application data and send it over to the destination where demultiplexing is used in order to determine which application the message is intended for.

This layer is also responsible for error and flow control. Flow control describes the process of ensuring that the data was received using ACK signals and responses. Error control refers to the process of ensuring that the data is in the same format as it was when it was initially sent, and it has not been malformed during transmission.

Protocols for this layer are TCP and UDP, or Transmission Control Protocol and User Datagram Protocol.

Network

The Network layer is where the IP protocol lives, which as we know is responsible for routing packets to the correct destination. This layer ensures the most efficient path to the destination is used based on transmission speed and the current load. This is where packets are created.

The IP protocol is used in this layer, along with ARP and ICMP

Data Link

The Data Link layer manages how the host device interface with the network adapter. Thus, the Data Link layer manages the communication between two devices *on the same network.*

When data is being sent, this layer takes packets and breaks them down into frames. These frames are then sent to the physical layer for transimission.

Protocols for this layer are DSL, LAN, SATNET, SONET, WiFi (802.11)

Physical

The Physical layer is made up of the physical equipment and infrastructure which facilitates the connection between two devices. This layer transmits data only as 1's and 0's, called signals, which is then interpreted on the receiving end to reconstruct the data by following the OSI model in reverse.

The Ethernet (802.3) protocol would live on this layer, since it is a physical connection between machines

Resources and Links

Pearson - OSI Model

Cloudflare - OSI Model

TCP / UDP

TCP

Transmission Control Protocol

In TCP, the client must first contact the server and request to connect to socket which the server has been configured to listen on. The server accepts the connection and creates a new socket for communicating with the client, so there can simultaneously be several connections at once. In TCP, there is error detection and data is received in the order it is sent. This error detection comes at the cost of speed, when compared to UDP.

TCP connections are established with the following steps

1. **Bind** socket to listen on server
2. **Create** request from client to connect to socket
3. **Accept** connection request and provide the server with a unique socket for communicating with this new connection

A bytestream is transmitted over a connection oriented channel and is a constant stream of data from a source to a destination. Since this happens on a connection oriented channel, bytestreams are constant and are only lost if the connection is closed.

A few protocols which use TCP connections are HTTP, HTTPS, SMTP, POP, and FTP.

TCP SSL Handshake

First, we should distinguish between a SSL *session* and *connection*.

A session can have multiple connections at any given time. SSL sessions are cached by the browser, typically until the browser is closed entirely but this may vary depending on browser and configurations. SSL sessions are also remembered by the server itself, which is also configurable on the back-end of the server but typically these sessions can last anywhere from 10 minutes to 10 hours. When a session is established, *a master secret is created for the client, along with two random values - one for the client from the server, and one for the server from the client.*

When a new connection is established within a valid session, *new symmetric keys are established for-each connection*. This is because the symmetric keys are derived from the master secret and the random values provided by both the client and server. Using these three values, symmetric keys are created for-each connection under any given session. Typically, a server will close connections after 1-2 minutes of inactivity and thus a new handshake will be required in order to establish a new connection.

Symmetric keys are only stored in RAM. This means if you shut down your device you can guarentee that a new handshake will occur the next time a HTTPS request is made to the server.

The steps required for a successful TCP SSL Handshake are seen below

1. Client sends `hello` message to the server, including a TLS version supported by the client, the cipher suites supported by the client, and a string of random bytes called the `client random`
2. The server responds with a `hello` message sent to the client, including the SSL certificate, the cipher suite used by the server, and a random string of bytes called the `server random`
3. The client verifies the SSL certificate with the Certificate Autority that issued the certificate, confirming the server is who it claims to be
4. The client sends one final random string of bytes called the `master secret`, which is encrypted using the server's public key which was provided with the SSL certificate from the previous steps
5. The server decrypts the `master secret` string using it's private key
6. Both the server and the client produce a `session key` using the `client random`, `server random`, and `master secret`. Since both the client and the server produce their own `session key` from the same ingredients, they both arrive at the same result and have matching `session keys`
7. The client sends a `finished` message encrypted with it's session key
8. The server sends a `finished` message encrypted with it's session key
9. Symmetric encryption has been established, and the handshake has completed. Secure communication can continue between the client and the server.

It is worth noting that depending on the cipher suite selected by the server in step 2, asymmetric encryption may or may not be used in the steps above. This means that steps 4-6 can vary depending on the cipher suite.

UDP

User Datagram Protocol

In UDP, there is no connection between a client and a server, packets may be sent out of order, and packets may be lost. The data keeps transmitting regardless, the sender using an IP and port which the recipient can derive from the datagram. UDP does not check for errors, and as a result has a faster speed than TCP connections. Since UDP is conectionless, there are no steps to establish a UDP connection.

UDP would be preferred in situations where data transfer does not stop if a segment is lost, like streaming a video or playing an online multiplayer game

A datagram is transmitted over a connectionless communication channel and represents just one part of a message being sent. Datagrams can be lost during transmission and resent.

Resources and Links

Cloudflare - TLS SSL Handshake

SSL.com - SSL/TLS Handshake

StackOverflow - SSL Session & Connection