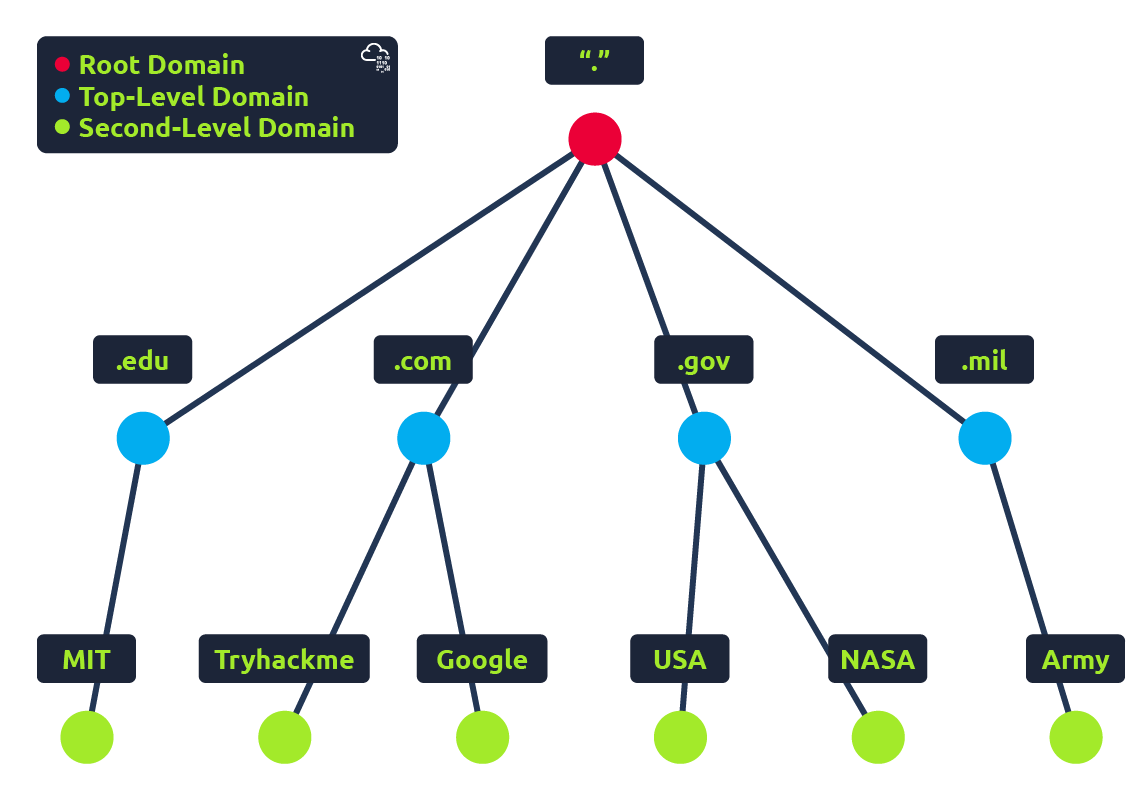
**What is DNS?**

DNS (Domain Name System) provides a simple way for us to communicate with devices on the internet without remembering complex numbers. Much like every house has a unique address for sending mail directly to it, every computer on the internet has its own unique address to communicate with it called an IP address. An IP address looks like the following 104.26.10.229, 4 sets of digits ranging from 0 - 255 separated by a period. When you want to visit a website, it's not exactly convenient to remember this complicated set of numbers, and that's where DNS can help. So instead of remembering 104.26.10.229, you can remember tryhackme.com instead.

**Domain Hierarchy**



**TLD (Top-Level Domain)**

A TLD is the most righthand part of a domain name. So, for example, the tryhackme.com TLD is **.com**. There are two types of TLD, gTLD (Generic Top Level) and ccTLD (Country Code Top Level Domain). Historically a gTLD was meant to tell the user the domain name's purpose; for example, a .com would be for commercial purposes, .org for an organisation, .edu for education and .gov for government. And a ccTLD was used for geographical purposes, for example, .ca for sites based in Canada, .co.uk for sites based in the United Kingdom and so on. Due to such demand, there is an influx of new gTLDs ranging from .online , .club , .website , .biz and so many more. For a full list of over 2000 TLDs [click here](https://data.iana.org/TLD/tlds-alpha-by-domain.txt).

**Second-Level Domain**

Taking tryhackme.com as an example, the .com part is the TLD, and tryhackme is the Second Level Domain. When registering a domain name, the second-level domain is limited to 63 characters + the TLD and can only use a-z 0-9 and hyphens (cannot start or end with hyphens or have consecutive hyphens).

**Subdomain**

A subdomain sits on the left-hand side of the Second-Level Domain using a period to separate it; for example, in the name admin.tryhackme.com the admin part is the subdomain. A subdomain name has the same creation restrictions as a Second-Level Domain, being limited to 63 characters and can only use a-z 0-9 and hyphens (cannot start or end with hyphens or have consecutive hyphens). You can use multiple subdomains split with periods to create longer names, such as jupiter.servers.tryhackme.com. But the length must be kept to 253 characters or less. There is no limit to the number of subdomains you can create for your domain name.

**DNS Record Types**

DNS isn't just for websites though, and multiple types of DNS record exist. We'll go over some of the most common ones that you're likely to come across.

**A Record**

These records resolve to IPv4 addresses, for example 104.26.10.229

**AAAA Record**

These records resolve to IPv6 addresses, for example 2606:4700:20::681a:be5

**CNAME Record**

These records resolve to another domain name, for example, TryHackMe's online shop has the subdomain name store.tryhackme.com which returns a CNAME record shops.shopify.com. Another DNS request would then be made to shops.shopify.com to work out the IP address.

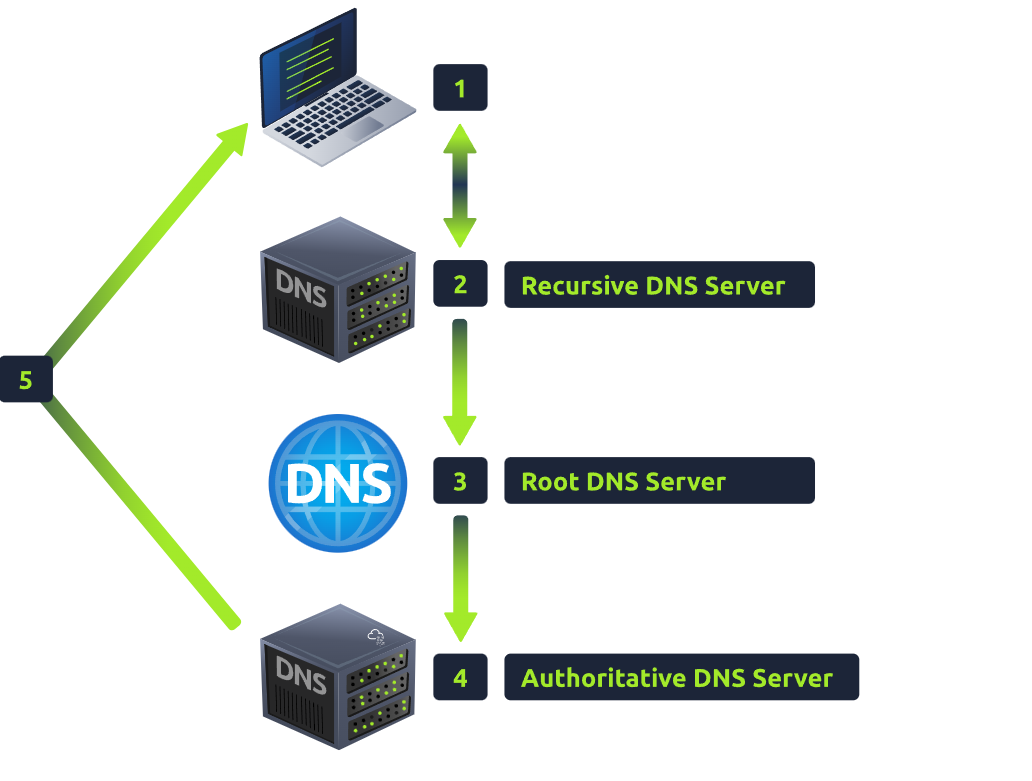
**MX Record**

These records resolve to the address of the servers that handle the email for the domain you are querying, for example an MX record response for tryhackme.com would look something like alt1.aspmx.l.google.com. These records also come with a priority flag. This tells the client in which order to try the servers, this is perfect for if the main server goes down and email needs to be sent to a backup server.

**TXT Record**

TXT records are free text fields where any text-based data can be stored. TXT records have multiple uses, but some common ones can be to list servers that have the authority to send an email on behalf of the domain (this can help in the battle against spam and spoofed email). They can also be used to verify ownership of the domain name when signing up for third party services.

**What happens when you make a DNS request**



1. When you request a domain name, your computer first checks its local cache to see if you've previously looked up the address recently; if not, a request to your Recursive DNS Server will be made.
2. A Recursive DNS Server is usually provided by your ISP, but you can also choose your own. This server also has a local cache of recently looked up domain names. If a result is found locally, this is sent back to your computer, and your request ends here (this is common for popular and heavily requested services such as Google, Facebook, Twitter). If the request cannot be found locally, a journey begins to find the correct answer, starting with the internet's root DNS servers.
3. The root servers act as the DNS backbone of the internet; their job is to redirect you to the correct Top Level Domain Server, depending on your request. If, for example, you request [www.tryhackme.com](http://www.tryhackme.com/), the root server will recognise the Top Level Domain of .com and refer you to the correct TLD server that deals with .com addresses.
4. The TLD server holds records for where to find the authoritative server to answer the DNS request. The authoritative server is often also known as the nameserver for the domain. For example, the name server for [tryhackme.com](http://tryhackme.com/) is [kip.ns.cloudflare.com](http://kip.ns.cloudflare.com/) and [uma.ns.cloudflare.com](http://uma.ns.cloudflare.com/). You'll often find multiple nameservers for a domain name to act as a backup in case one goes down.
5. An authoritative DNS server is the server that is responsible for storing the DNS records for a particular domain name and where any updates to your domain name DNS records would be made. Depending on the record type, the DNS record is then sent back to the Recursive DNS Server, where a local copy will be cached for future requests and then relayed back to the original client that made the request. DNS records all come with a TTL (Time To Live) value. This value is a number represented in seconds that the response should be saved for locally until you have to look it up again. Caching saves on having to make a DNS request every time you communicate with a server.

**What is HTTP? (HyperText Transfer Protocol)**

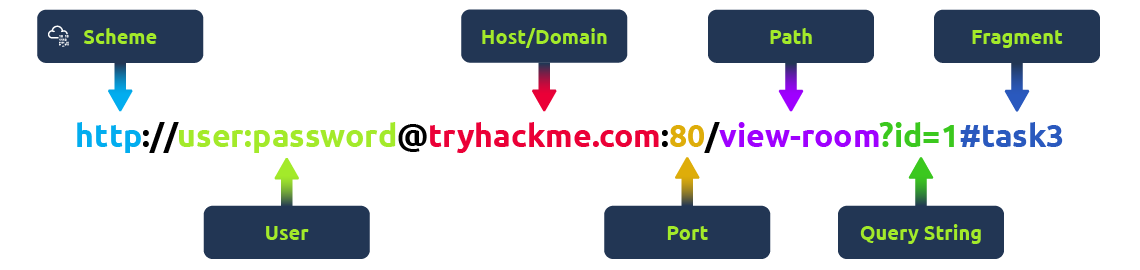
HTTP is what's used whenever you view a website, developed by Tim Berners-Lee and his team between 1989-1991. HTTP is the set of rules used for communicating with web servers for the transmitting of webpage data, whether that is HTML, Images, Videos, etc.

**What is HTTPS? (HyperText Transfer Protocol Secure)**

HTTPS is the secure version of HTTP. HTTPS data is encrypted so it not only stops people from seeing the data you are receiving and sending, but it also gives you assurances that you're talking to the correct web server and not something impersonating it.

**What is a URL? (Uniform Resource Locator)**

If you’ve used the internet, you’ve used a URL before. A URL is predominantly an instruction on how to access a resource on the internet. The below image shows what a URL looks like with all of its features (it does not use all features in every request).



**Scheme:** This instructs on what protocol to use for accessing the resource such as HTTP, HTTPS, FTP (File Transfer Protocol).

**User:** Some services require authentication to log in, you can put a username and password into the URL to log in.

**Host:** The domain name or IP address of the server you wish to access.

**Port:** The Port that you are going to connect to, usually 80 for HTTP and 443 for HTTPS, but this can be hosted on any port between 1 - 65535.

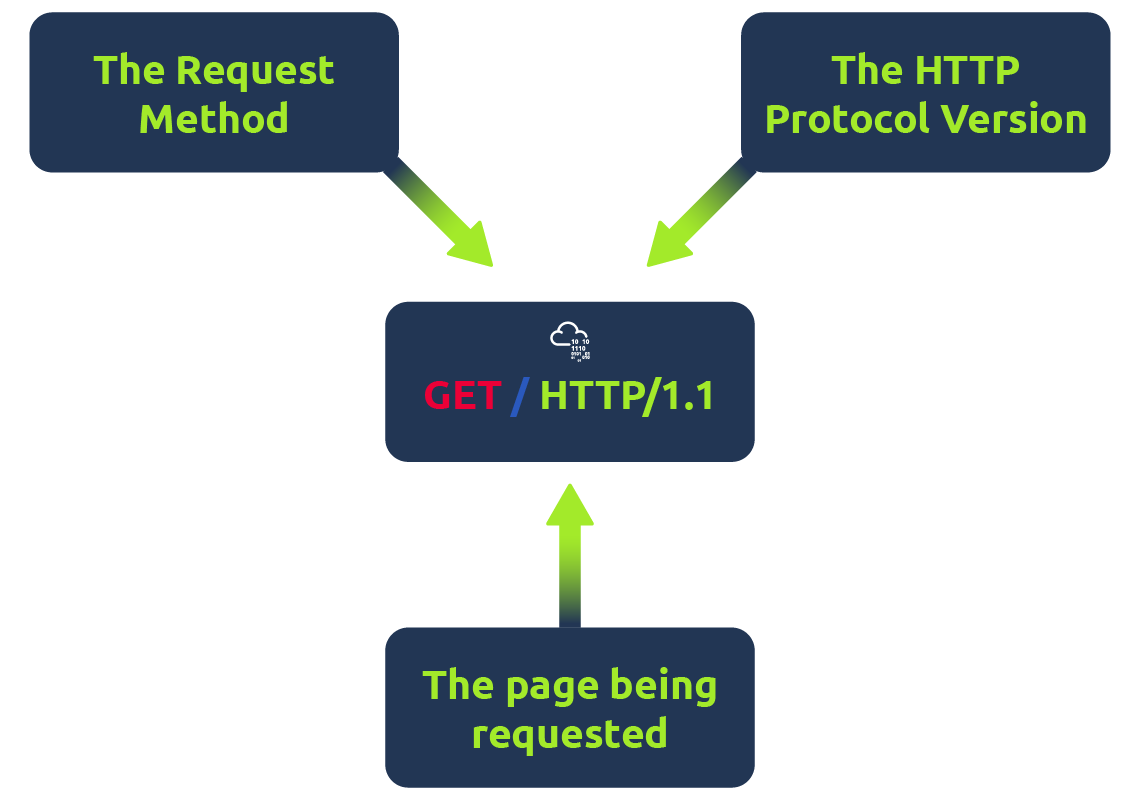
**Path:** The file name or location of the resource you are trying to access.

**Query String:** Extra bits of information that can be sent to the requested path. For example, /blog?**id=1**would tell the blog path that you wish to receive the blog article with the id of 1.

**Fragment:**This is a reference to a location on the actual page requested. This is commonly used for pages with long content and can have a certain part of the page directly linked to it, so it is viewable to the user as soon as they access the page.

**Making a Request**

It's possible to make a request to a web server with just one line "**GET / HTTP/1.1**"



But for a much richer web experience, you’ll need to send other data as well. This other data is sent in what is called headers, where headers contain extra information to give to the web server you’re communicating with, but we’ll go more into this in the Header task.

**Example Request:**

GET / HTTP/1.1

Host: tryhackme.com

User-Agent: Mozilla/5.0 Firefox/87.0

Referer: https://tryhackme.com/

To breakdown each line of this request:

**Line 1:** This request is sending the GET method ( more on this in the HTTP Methods task ), request the home page with / and telling the web server we are using HTTP protocol version 1.1.

**Line 2:** We tell the web server we want the website tryhackme.com

**Line 3:** We tell the web server we are using the Firefox version 87 Browser

**Line 4:** We are telling the web server that the web page that referred us to this one is [https://tryhackme.com](https://tryhackme.com/)

**Line 5:**HTTP requests always end with a blank line to inform the web server that the request has finished.

**Example Response:**

HTTP/1.1 200 OK

Server: nginx/1.15.8

Date: Fri, 09 Apr 2021 13:34:03 GMT

Content-Type: text/html

Content-Length: 98

<html>

<head>

<title>TryHackMe</title>

</head>

<body>

Welcome To TryHackMe.com

</body>

</html>

To breakdown each line of the response:

**Line 1:**HTTP 1.1 is the version of the HTTP protocol the server is using and then followed by the HTTP Status Code in this case "200 Ok" which tells us the request has completed successfully.

**Line 2:** This tells us the web server software and version number.

**Line 3:** The current date, time and timezone of the web server.

**Line 4:** The Content-Type header tells the client what sort of information is going to be sent, such as HTML, images, videos, pdf, XML.

**Line 5:** Content-Length tells the client how long the response is, this way we can confirm no data is missing.

**Line 6:** HTTP response contains a blank line to confirm the end of the HTTP response.

**Lines 7-14:** The information that has been requested, in this instance the homepage.

HTTP methods are a way for the client to show their intended action when making an HTTP request. There are a lot of HTTP methods but we'll cover the most common ones, although mostly you'll deal with the GET and POST method.

**GET Request**

This is used for getting information from a web server.

**POST Request**

This is used for submitting data to the web server and potentially creating new records

**PUT Request**

This is used for submitting data to a web server to update information

**DELETE Request**

This is used for deleting information/records from a web server.

**HTTP Status Codes:**

In the previous task, you learnt that when a HTTP server responds, the first line always contains a status code informing the client of the outcome of their request and also potentially how to handle it. These status codes can be broken down into 5 different ranges:

|  |
| --- |
| **100-199 - Information Response** |
| **200-299 - Success** |
| **300-399 - Redirection** |
| **400-499 - Client Errors** |
| **500-599 - Server Errors** |

**Common HTTP Status Codes:**

There are a lot of different HTTP status codes and that's not including the fact that applications can even define their own, we'll go over the most common HTTP responses you are likely to come across:

|  |
| --- |
| **200 - OK** |
| **201 - Created** |
| **301 - Moved Permanently** |
| **302 - Found** |
| **400 - Bad Request** |
| **401 - Not Authorised** |
| **403 - Forbidden** |
| **405 - Method Not Allowed** |
| **404 - Page Not Found** |
| **500 - Internal Service Error** |
| **503 - Service Unavailable** |

**Common Request Headers**

﻿These are headers that are sent from the client (usually your browser) to the server.

**Host:** Some web servers host multiple websites so by providing the host headers you can tell it which one you require, otherwise you'll just receive the default website for the server.

**User-Agent:** This is your browser software and version number, telling the web server your browser software helps it format the website properly for your browser and also some elements of HTML, JavaScript and CSS are only available in certain browsers.

**Content-Length:** When sending data to a web server such as in a form, the content length tells the web server how much data to expect in the web request. This way the server can ensure it isn't missing any data.

**Accept-Encoding:** Tells the web server what types of compression methods the browser supports so the data can be made smaller for transmitting over the internet.

**Cookie:** Data sent to the server to help remember your information (see cookies task for more information).

**Common Response Headers**

These are the headers that are returned to the client from the server after a request.

**Set-Cookie:** Information to store which gets sent back to the web server on each request (see cookies task for more information).

**Cache-Control:** How long to store the content of the response in the browser's cache before it requests it again.

**Content-Type:** This tells the client what type of data is being returned, i.e., HTML, CSS, JavaScript, Images, PDF, Video, etc. Using the content-type header the browser then knows how to process the data.

**Content-Encoding:**What method has been used to compress the data to make it smaller when sending it over the internet.

**Cookie :**

You've probably heard of cookies before, they're just a small piece of data that is stored on your computer. Cookies are saved when you receive a "Set-Cookie" header from a web server. Then every further request you make, you'll send the cookie data back to the web server. Because HTTP is stateless (doesn't keep track of your previous requests), cookies can be used to remind the web server who you are, some personal settings for the website or whether you've been to the website before. Let's take a look at this as an example HTTP request:



Cookies can be used for many purposes but are most commonly used for website authentication. The cookie value won't usually be a clear-text string where you can see the password, but a token (unique secret code that isn't easily humanly guessable).

**Security Headers:**

**Content Security Policy (CSP)**: Helps **prevent XSS attacks** by

controlling which sources of content are allowed to be loaded on a web

page.

**Strict-Transport-Security (HSTS):** Enforces the use of **HTTPS for**

**communication** with the server and prevents the browser from making

insecure connections.

**X-Frame-Options:** Prevents a web page from being **displayed in a frame**

**or iframe on another site**, mitigating clickjacking attacks.

**X-XSS-Protection:** Enables the **browser's built-in XSS protection** mechanism to block or sanitize reflected XSS attacks.

**X-Content-Type-Options:** Prevents browsers from **interpreting files as different MIME types** than what is specified in the Content-Type header, preventing MIME sniffing attacks.

**Cross-Origin-Resource-Policy (CORP):** Controls how resources on a web page are shared with other **origins, helping to prevent data leakage.**

**Cache-Control**: no-store, no-cache: Directives that control caching behavior to prevent storage of sensitive data in cache.

**Feature-Policy**: geolocation, camera, microphone, etc.:

Individual directives within Feature-Policy can control specific browser features, reducing attack surface.

**Content-Disposition**:

Specifies whether a browser should display a file inline or offer it as a download, helping to prevent unwanted execution of files.

**Access-Control-Allow-Origin**: Controls which origins are allowed to make cross-origin requests to your site, reducing the risk of unauthorized data access.

**X-Download-Options:** Prevents Internet Explorer from executing downloads that could potentially be unsafe.

**Cookie ::**

**Secure Flag**: When set, the cookie will only be transmitted over HTTPS connections, ensuring that the cookie's data is encrypted during transmission.

**HttpOnly Flag**: This flag prevents JavaScript from accessing the cookie, which helps mitigate cross-site scripting (XSS) attacks. It ensures that sensitive information within the cookie can only be accessed by the server.

**SameSite Flag**: This flag controls when a cookie should be sent by the browser. It can have three values: "Strict," "Lax," and "None." The "Strict" value restricts the cookie to be sent only in same-site requests, "Lax" allows cookies to be sent with top-level navigations (like clicking a link), and "None" allows cross-site requests to include the cookie. This flag helps prevent cross-site request forgery (CSRF) attacks.