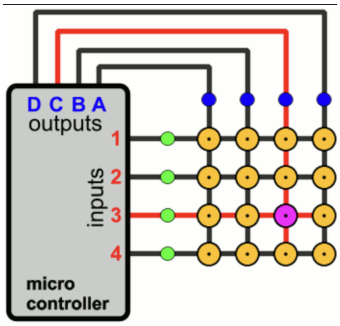
What happens when you enter a URL into a web browser and hit enter?

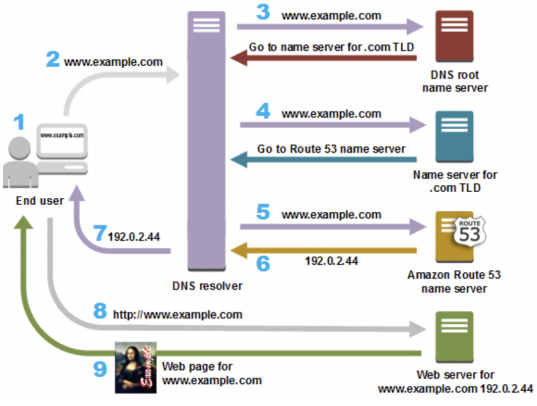
Written by Anthony Terrano
Network Engineer/Systems Administrator

Firstly, when you press the "Enter" key on a keyboard, a circuit is completed which sends an electrical signal to the keyboard's microcontroller. The microcontroller then processes this signal and sends a corresponding signal to the computer's Central Processing Unit (CPU) indicating that the "Enter" key has been pressed.



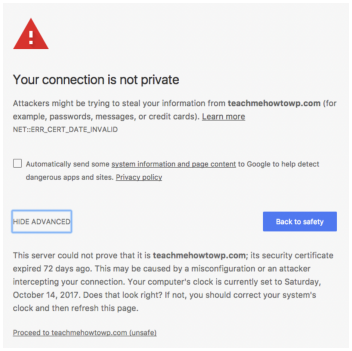
The keyboard's circuitry includes a key matrix which is a grid of rows and columns, with each key corresponding to a unique intersection of a row and a column. When you press a key, it completes a circuit between the row and column corresponding to that key (*Seen in the figure to the right)*, sending an electrical signal to the keyboard's microcontroller. The microcontroller then translates that signal into a corresponding scancode or character code, as an example, intersection (C,3) which is sent to the computer's CPU via the keyboard's interface via USB, or if you are on a laptop, through wiring in the motherboard. The CPU then interprets (C,3) as the "Enter" key being pressed, and sends those commands up to your operating system and creates an event.

When you press the "Enter" key in a web browser, the browser receives the event generated by the operating system and the browser's rendering engine handles that event. The browser rendering engine listens for the "keydown" event, which is triggered when a key is pressed down. The browser then sends an event to the webpage indicating that the "Enter" key has been pressed. The webpage can listen for this event using JavaScript to perform some action, such as performing a search. For example, if you are on a webpage with a search box, pressing "Enter" will trigger the "Submit" event for the search form and the browser will send a DNS query immediately following you hitting the “enter” key.



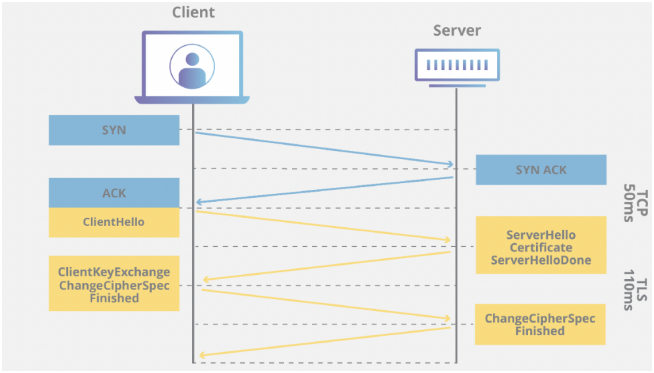
After the browser's event handler interprets the enter key and reads the URL (Uniform Resource Locator) your browser uses a DNS (Domain Name System) server to resolve the domain name in the URL to an IP (Internet Protocol) address. This process first starts by checking your browser's local DNS cache to see if it has already visited the website and has a copy of it. If it is not cached, the browser sends a DNS query to a DNS server. The DNS server responds with an IP address associated with the domain name. If no result is found in the DNS resolver, then it asks the name server listed above the first DNS server in the hierarchy, until it finds a result. Once it finds the correct IP address, it

then caches that address locally and the browser initiates a TCP connection to the web server at the IP address it obtained.



The browser then sends an HTTP (HyperText Transfer Protocol) request to the web server, including the domain name, requested resource, and any additional information, such as cookies or headers. The web server responds with an HTTP response, including the requested resource, HTTP headers, and any additional information, such as cookies or data. The browser receives the HTTP response and renders the webpage, including any HTML, CSS, JavaScript, or other content. In most modern browsers, you would get a security warning, as any data that transmits to and from the website this way is not encrypted and can be vulnerable. To fix this, engineers implement a more secure protocol called HTTPS.

When you type in a URL in your web browser and the website is secured with HTTPS (HyperText Transfer Protocol Secure), the process is similar to an HTTP request as discussed earlier, but with additional steps for establishing a secure connection between the browser and server. Firstly, the web browser sends a DNS query to a DNS server to resolve the domain name in the URL to obtain an IP address. The browser initiates a TCP connection to the web server. A TCP handshake starts with the client sending a SYN packet to the server to initiate the connection. The server responds with a SYN-ACK packet to acknowledge the request and initiate a connection. The client responds with an ACK packet to confirm that the connection has been established. Then, the browser sends a TLS (Transport Layer Security) handshake message to the web server to initiate a secure connection, commonly called the “ClientHello”. The message includes the supported TLS versions, cipher suites, and random values. The web server responds with a TLS handshake response message that includes its SSL/TLS certificate, which includes its public key, domain name, and other information, called the “ServerHello”. The browser verifies the SSL/TLS certificate to ensure it is issued by a trusted Certificate Authority (CA) and that the domain name matches the server's domain name. The browser sends a "Finished" message to indicate that it has completed its part of the handshake. The server sends a "Finished" message to indicate that it has completed its part of the handshake.



The browser and server now have a shared symmetric key, which is used to encrypt and decrypt all data exchanged between them during the HTTPS session. From there, the browser sends an HTTPS request to the server, including the domain name, requested resource (such as a web page), and any additional information, such as cookies or headers. The web server

responds, and once the HTTPS response has been received, the browser and server close the HTTPS session.

The encryption type is defined in HTTPS by use of a cipher suite, which in more detail, is a combination of cryptographic algorithms that are used to secure a connection between a client and server. When establishing a secure connection using protocols such as HTTPS, the client and server agree on a cipher suite that will be used to encrypt and decrypt data exchanged during the session.

On average, a cipher suite will consist of all or more of the following components:

1. Key Exchange Algorithm: This algorithm is used to securely exchange cryptographic keys between the client and server. Examples include Diffie-Hellman and Elliptic Curve Diffie-Hellman (ECDH).
2. Encryption Algorithm: This algorithm is used to encrypt data transmitted between the client and server. Examples include Advanced Encryption Standard (AES), Triple Data Encryption Standard (3DES), and RC4.
3. Message Authentication Code (MAC) Algorithm: This algorithm is used to ensure the integrity of data transmitted between the client and server. Examples include Hash-based Message Authentication Code (HMAC) and Cipher-based Message Authentication Code (CMAC).
4. Protocol Version: This specifies the version of the protocol being used, such as TLS 1.2 or TLS 1.3.

Arguably, most important to the cipher suit is the key exchange. When you do this over the internet you have to take into account that an encryption agreement is established in public, so how do we do this without giving away any of the secrets? Let's take a closer look at the Diffie-Hellman key exchange algorithm. The Diffie-Hellman key exchange is a method for two parties to agree on a shared secret key over an insecure communication channel, without ever exchanging the key directly. The method is based on the difficulty of computing discrete logarithms in a finite field, and it works as follows:

1. Alice and Bob agree on a large prime number "p" and a generator "g", where g is a primitive root modulo p.
2. Alice chooses a random secret integer a and computes A = g^a mod p. She sends A to Bob.
3. Bob chooses a random secret integer b and computes B = g^b mod p. He sends B to Alice.
4. Alice computes s = B^a mod p, which is equal to (g^b mod p)^a mod p.
5. Bob computes s = A^b mod p, which is equal to (g^a mod p)^b mod p.

To see how the Diffie-Hellman key exchange works mathematically, let's look at an example:

1. Alice and Bob agree on a large prime number p = 23 and a generator g = 5.

2. Alice chooses a random secret integer a = 6 and computes A = g^a mod p = 5^6 mod 23 = 8. She sends A = 8 to Bob.
3. Bob chooses a random secret integer b = 15 and computes B = g^b mod p = 5^15 mod 23 = 19. He sends B = 19 to Alice.
4. Alice computes s = B^a mod p = 19^6 mod 23 = 2.
5. Bob computes s = A^b mod p = 8^15 mod 23 = 2.

Now Alice and Bob both have the same shared secret key s = 2, which they can use for symmetric encryption. Therefore, an eavesdropper who intercepts the messages exchanged between Alice and Bob would not be able to compute the shared secret key without solving the discrete logarithm problem.

At this point, the connection is established, and the client and server can exchange data using the agreed-upon cipher suite. The encryption and MAC algorithms specified in the cipher suite are used to ensure that the data exchanged between the client and server is secure and protected from interception or snooping, and the user may begin browsing the various web pages offered by the website hosting server.