Thomas Hart

CS 465 HW #8

The article discusses the specifics behind an HTTPS connection. Most commonly, this is done through a Transport Layer Security (TLS) protocol, which is a new and re-branded form of the original Secure Sockets Layer (SSL) protocol. TLS will wrap records of different types. For an HTTPS connection, it shows the first byte 0x16 to specify that this is a “handshake” record. It gives the TLS version and the length of the request. It then jumps into the “client hello” step.

The client hello consists of four sections: random, session ID, cipher suites, and server name extension. Random has a timestamp and a few other bytes that will be used later in the process. The session ID shows whether a handshake has happened recently enough to forgo the entire process. The cipher suites indicate the acceptable encryption algorithms the protocol will support to send the message. Finally, the server name extension says what website is trying to be accessed.

After the client hello has reached the server, the server sends back a response called the “server hello”, which consists of three sub-messages. It starts with the same TLS version sent in the client hello. This indicates that the server has agreed to use the TLS version requested. The server hello also has “random” and “session ID” sections, followed by the encryption algorithm it will use. The second sub-message is a long message, which is the website’s certificate the client can validate. The third and final sub-message is the server telling the client that the “server hello” is complete.

The next step is validating the certificate the server has sent back to the client. The article shows the case where this is done through RSA. The digital signature is decrypted, and since everything has the expected values – including the validation of the hash – the client can trust that the response is authentic. We know they are they only ones with private key ‘d’, and our communication with them will be safe. You can then repeat the process to verify that the certificate was signed by an appropriate third party.

Third party companies are used as a form of mutual trust. In other words, everyone has agreed that a certain third-party company is trustworthy to sign certificates, validating their authenticity. There is no proof of this trust. It is simply implicit. This seems like an issue, but there is no way around this problem. The entities needing validation must have some sort of trust chain.

There is one final check, which is to see if the name on the certificate matches the name of the website we are trying to reach. This prevents a man-in-the-middle attack. If it were someone else, the certificate would not indicate the website we are trying to reach, but something else. This is under the assumption that the third-party company really is trustworthy and hasn’t made a certificate that shows an entity is something other than what it is.

After validating some information about the website, the client has the public encryption exponent ‘e’ and modulus ‘n’. The client now needs to create a random secret key that they can send to the server securely. Nobody else can know it. This is done through whatever the server’s preferred algorithm was – in this case, RSA. This will require the client to not only send the generated secret key, but also add random padding to make it match the length of the modulus. This protects against weaknesses or stupid mistakes such as using the same secret key twice. The random padding will make figuring it out much more difficult.

After all this is done correctly, the client and server both have the “pre-master secret”. Amazon is now the one that needs to verify that the client is trustworthy. It takes the pre-master secret key and performs some hashing on it. This becomes the master key and is sent back to the client. Now that both the client and the server have the master key, a final message is sent to verify that the handshake was successful and that both sides have the master key. The TLS protocol establishes a secure pipeline for data to be sent back and forth between client and server. The messages can only be interpreted through decrypting the ciphertext using the secret keys exchanged by the two entities.

Regular HTTP traffic can now be sent while the TLS layer encrypts and verifies the integrity of messages. The connection will stay open until either the client or the server sends a “closure alert”, and the connection is closed. If there is an attempt to reestablish a connection shortly after, the session ID mentioned before can allow the client and server to forgo the handshake and still make the secure connection. Otherwise, the entire handshake process will happen all over again.