Encrypting a Website Connection with HTTPS

Note: Since encrypting the connection cannot be done by modifying the source code, I wanted to write a detailed explanation of how HTTPS works and how it can be deployed. I consulted Mr. Ian Butler prior to this, and he said I can write this document to get credit.

The website is vulnerable to a Man-in-the-Middle attack (MITM). An MITM attack occurs when the connection between the server and the client is not encrypted. In this case, data is transmitted in plain text, and an intruder on the same network can jump in and steal sensitive data such as usernames and passwords.

I exploited the vulnerability by using WireShark to capture the packet that included the username and password. First, open the website on one computer. In my case I ran a virtual machine on VMware. Open Wireshark on another computer. In my case it's the macOS on which the virtual machine is running. Choose the network the targeted computer is on. In my case it's the bridging network between the virtual machine and the host OS. Now Wireshark starts capturing packets on the network.Register on the website. In my case I entered the username "test@test.com" and the password "test". On Wireshark, click "Export Objects". Now you can see all the objects that can be extracted from the packets. Go to the log-in page of the website. Type in the registered username and password. While logging in, the computer transmits the file "login.json" to the website, which Wireshark captures in a packet. It's shown as an exportable object on the list. Save the highlighted file as "login.json". Open the file in a code editor, and you can see the username and password.

A solution to the vulnerability is to encrypt the website's connection via HTTPS. A secure HTTP connection like HTTPS is made possible by a protocol called Transport Layer Security

(TLS). TLS uses both Public Key Cryptography and Symmetric Key Cryptography to encrypt the connection. Public Key Cryptography only happens once per session. The goal is for the client and the server to agree upon a shared secret key so that they can use Symmetric Key Cryptography for the rest of the sessions, which is much less time-consuming. The shared secret key is generated by a Diffie-Hellman (DH) key exchange.

For example, Alex and John are two parties establishing an encrypted connection. Both of them have unique private keys, which are over a hundred digits long each. They both generate a mixture of their keys via a math formula and send them publicly. They agree on a root number and a large prime number. Alex's mixture would be the root raised to the power of his private key, mod the prime number. John's mixture would be the root raised to the power of John's private key, mod the prime number. Now they send the numbers to each other publicly. Once Alex gets John's mixture number, he calculates John's mixture raised to Alex's private key, mod the prime number. The same thing goes for John. The formula results in the same number for both John and Alex, and that number becomes the shared key for the session. With a shared key plus an agreed-upon collection of encryption algorithms, the client and server can then use Symmetric Key Encryption for the rest of the sessions, which takes much less time.

Another part of the solution is that Alex and John need to know they are talking to the real other person. Even though the data is encrypted, an intruder can still disguise as the other party and receive the data. This is where public key certificates come in. Public key certificates are signed by trusted Certificate Authorities (CA). CAs verify each website's identity and issue certificates accordingly. The certificates inform the user that on the other side of the Internet is the actual website they want to use. For example, Google has a public key certificate signed by a

CA. Every time you visit Google, Google returns the certificate that shows that they are authenticated by a CA. If an intruder generates their own certificate and claim they are Google, your browser will return a red flag.

The website provided for this project does not have an encrypted connection. To deploy an encrypted connection, a public key certificate need to be acquired from a CA. Popular CAs in the US include VeriSign and DigiCert. Once a certificate is acquired, users visiting the website will be assured that they are actually exchanging data with the true server, as opposed to some intruder's scam. Afterwards, deploy Transport Layer Security (TLS) and install cipher suites such as AES 256 CBC SHA.

Now the website has an encrypted connection, and users will see a green padlock on the browser every time they visit. There is no chance an attacker can bypass this fix. From top to bottom, the transmission is secured by a public key certificate, cipher suites and TLS key exchange. There is no way data can be compromised.

References:

https://blog.hartleybrody.com/https-certificates/

https://github.com/ssllabs/research/wiki/SSL-and-TLS-Deployment-Best-Practices