Working with SSL Certificates

SSL à “Secure Sockets Layer”

TLS à “Transport Layer Security”

SSL is called TLS now, SSL is historical, but the name is still commonly used. Think of TLS as an upgrade over SSL.

SSL had a lot of security issues, and TLS closed many of those security holes.

What are these certificates used for?

1st question: What are SSL certificates?

Let’s say I am entering a password on a website or entering credit card details on a website, how can I be sure that where I am sending this info is legitimate? And not to a malicious clone of the website or how can I make sure that no one is stealing the information that I am sending through the network?

Certificates solve both issues.

They authenticate the website, and they encrypt network traffic between the user and the website.  
Authentication in this context means that the website can cryptographically prove that a browser is legitimate. This proves that the website is who it says it is and not some clone of it.  
The certificate is also used to make the connection between the user and the website private. All data exchanged between the user and the website is encrypted.

On Linux, Openssl is the utility usually used to create and manage TLS certificates. Despite the name, one would use openssl to create TLS certificates.

For this exam, what interests us is the creation of X.509 certificates, these are the certificates that we can use on websites to do the authentication and encryption mentioned earlier in my notes.

Commands:  
~$ openssl à this command will show you the numerous sub-commands that can be used with this utility.

Each sub-command has its own manual. To see it type:  
~$ man openssl TAB+TAB (on keyboard) don’t type space after openssl

Add the name of the subcommand after man openssl to check each man page.

What is a Certificate Signing Request (CSR)?

The req sub-command deals with CSRs.

This is because the digital certs used to secure website traffic are not enough on their own.

Example.com can use a certificate to secure web traffic between a user and a website and that is not an issue. But, when a user visits example.com, their browser needs to also trust the certificate. It needs a way to be sure the certificate is legitimate and not some fake created by a hacker. So how can it be sure? By checking if something called a certificate authority signed example.com’s certificate.   
I can send a certificate that I generated locally to some company like google, then they would us a special private key and sign my certificate, once that certificate is signed, any browser can then check the signature and notice that google validated it so its legit.

Generating a Key and a CSR

Certificates are normally used in conjunction with a secret key.

Command to generate both at the same time:

~$: openssl req -newkey rsa:2048 -keyout key.pem -out req.pem

-newkey rsa:2048 à lets us instruct openssl to generate a key of this type (rsa) with 2048 bits.

-keyout key.pem à tells the utility to save the private key in a file called key.pem (private key)

-out req.pem à saves my certificate signing request (CSR) to a file called req.pem (CSR)

You can run cat key.pem to analyze the content. You can also run cat on req.pem to analyze that content as well.

In the real world, I would send the CSR to a CA, they would check and verify my details, we would prove our identity, pay a fee (there is an open-source solution that’s free) and get a certificate.

How to generate a self-signed certificate

When we need the whole world to trust and validate our certificate, we need the services of a CA. But other scenarios exist.  
For example, I might want to generate a certificate for internal use in my office network. And I can configure all my company’s computers to trust my local certificates.

In this situation, we can perform self-signing a certificate.

So, we can skip generating a CSR and sending it to an authority. I can generate the final, signed certificate myself.

-x509 à generates an x509 type of certificate, instead of generating a CSR.

-noenc à Don’t ask me for a password to encrypt this key (not recommended in a real scenario)

-newkey rsa:4096 à tells openssl to use the RSA algorithm for this key & use 4096 bits for it. (default is RSA:2048)

-days 365 à makes cert valid for a year

-keyout myprivate.key à saves the private key to a file called myprivate.key

-out mycertificate.crt à saves the certificate to a file called mycertificate.crt.

To analyze the certificate, I can use the following command:

~$: openssl x509 -in mycertificate.crt -text

There will be a lot of text in the output, so scroll up to see the info that will interest me.