1.3 HTTPS

When the commercials boast about ''secure web servers'', they normally refer to web servers capable of doing encrypted communication. As this book will show you, it takes far more than encryption to make a web server secure, but encryption plays an important role (You'll find a short introduction to cryptology in Section 6.1. Consider reading it first if you're not familiar with words like ''encryption'', ''hash'' and ''certificate'').

In a web setting, encryption usually means HTTPS. Using simple terms, HTTPS may be described as good, old HTTP communicated over an encrypted channel. The encrypted channel is provided by a protocol named Secure Socket Layer (SSL) [34], or by its successor Transport Layer Security (TLS) [35, 36]. It is important to realize that the encryption only protects the network connection between the client and the server. An attacker may still attack both the server and the client, but he will have a hard time attacking the communication channel between them.

When SSL/TLS is used, the client and the server start by performing a *handshake*. The following is done as part of that handshake (we leave the hairy details out):

- The client and the server agree on what crypto- and hashing algorithms to use.
- The client receives a certificate from the server and validates it.

- Both agree on a symmetric encryption key.
- Encrypted communication starts.

The handshake may also include a client certificate to let the server authenticate the client, but that step is optional. After the handshake is done, control is passed to the original handler, who now talks plain HTTP over the encrypted channel.

If everything works as expected, HTTPS makes it impossible for someone to listen to traffic in order to extract secrets. People may still sniff packets, but the packets contain seemingly random data. HTTPS thus protects against packet sniffing (Appendix B).

If everything works as expected, HTTPS protects against what is known as *man in the middle* attacks (MITM), too. With MITM, the attacker somehow fools the victim's computer into connecting to him rather than to, say, the bank. The attacker then connects to the bank on behalf of the victim, and effectively sits between the communicating parties, passing messages back and forth. He may thus both listen to and modify the communication. When HTTPS is used, the clients will always verify the server's certificate. Due to the way certificates are generated, the man in the middle will not be able to create a fake but valid certificate for the web site. Any MITM attempts will thus be detected. If everything works as expected, that is. (See Appendix B on page 193 for more on MITM.)

Advanced

Why would the attacker need to create a server certificate? Why not just pass the real server certificate along? Inside the certificate is a public key. The corresponding private key is only to be found on the server. As part of the handshake, the server uses the private key to sign some information. The client will use the public key in the certificate to verify that the signature was in fact made by the server's private key. The attacker doesn't know the server's private key, so he'll have to create a new key pair in order to fulfil the signing requirements. Then he will also need to make a new certificate to include his own public key, but he won't be able to sign the certificate with the key of a well-known CA (certification authority). He will have to sign the certificate himself, and browsers will thus complain about an unknown CA.

If you read between my lines, you may notice a certain lack of enthusiasm. And you're right. As I see it, HTTPS in real life doesn't always solve the

problems it is supposed to solve. To see what I mean, we have to understand a little bit more.

Recent versions of the SSL and TLS specifications are thought to be secure. The problems come not from the standards themselves, but from how they work in the real world. First of all, you have the users. When the browser pops up a window stating that "This certificate expired two months ago" or "This certificate is signed by an unknown certification authority", what will the average user do? Most likely, he won't understand anything, and he will click "Continue" in order to be able to buy what he intended to buy. The security is ruined by the end user. Or maybe by the browser who allowed that ignorant user to press "Continue" in the first place.

HTTPS certificates are tightly coupled to the domain names of the web sites they protect. If an attacker gets hold of a domain name that is similar to an actual site, such as `mega-bank.example.com` that has an extra hyphen compared to the real "Mega Bank" at `megabank.example.com`, he may trick a user into visiting his fake site, which typically holds a copy of the real site. If he succeeds, there need not be any warnings from the browser at all: either the attacker has bought a valid certificate for the domain he owns, or, if the user really has no clue, the attacker uses plain HTTP rather than HTTPS. Many users may fall for this kind of trick, as lots of people are unfamiliar with domain names and protocols. Just look at the search statistics at popular search engines. Example: The term "Yahoo" is one of the most popular searches on Fast's search engine `www.alltheweb.com`. Why don't people just enter `www.yahoo.com` in the address bar rather than adding an unnecessary search step?

Then you have the certification authorities (CA). Their job is to approve that a web site is actually the web site it seems to be. CAs provide web sites with certificates that are digitally signed by the CA. Every browser out there has a built-in catalog of CA certificates. These browser-built-in certificates are used to verify that server certificates are signed by a trusted CA. The catalog includes the CAs that your browser vendor has faith in, and the trust of your browser vendor will automatically be your trust, unless you modify the catalog. Most users don't. They don't even know that the built-in list of CAs has something to do with trust.

Now, can we trust the CAs? In 2001, someone tricked VeriSign, the largest and most well-known CA out there, into providing "Microsoft Corporation" certificates to someone who was not from Microsoft Corporation [37, 38]. The certificates were code signing certificates, not web server certificates, but the incident nevertheless demonstrates that even CAs do make mistakes.

If a CA does make a mistake, or if the web server certificate somehow gets compromised, what can be done? The certificate will have to be invalidated, or revoked, before its original expiry date. The only way to check for premature invalidation is to have the browsers check it on each SSL/TLS connection. The Online Certificate Status Protocol (OCSP) [39] has been invented for that particular purpose. With OCSP, the browser will connect to the CA for each certificate it receives, and ask if the certificate is still valid. Alternatively, the browser may periodically download a Certificate Revocation List (CRL) [40, 41] from the CAs, and check if a certificate is included in this list of withdrawn certificates. Several browsers support OCSP and CRLs, but as of this writing, none of them have the protocols activated in the default configuration. Browsers will thus continue to use compromised certificates until they expire by natural causes.

When the CAs who make their living by selling trust make mistakes, everyone else may make mistakes too. Most major browsers have had several bugs in their SSL or TLS implementations that make it possible to predict cryptographic keys, or bypass certificate validation. With such bugs present, HTTPS is of little use.

To conclude this paragraph, HTTPS provides good protection of the communication channel unless:

- The user neglects the warnings from the browser.
- The browser allows the user to neglect its warnings.
- The user falls for cheap domain name or protocol tricks played by an attacker.
- The CA may be tricked into giving out false certificates.
- The browser vendor trusts a CA that the user wouldn't trust.
- The browser (or server) has a buggy SSL/TLS implementation.
- The user's computer is controlled by an attacker.

As all of the above points have already been seen, you may realize that HTTPS is not a magic bullet that solves all problems in the real world. But don't get me wrong. HTTPS is the best widely available mechanism for securing web traffic at the moment. It makes it much harder to reach or modify secrets that pass across the network.