Security II - HTTPS

Stefano Calzavara

Università Ca' Foscari Venezia

March 13, 2020

Università
Ca'Foscari
Venezia

HTTP vs HTTPS

HTTP is a plaintext protocol

- no confidentiality and integrity against network attackers
- no authentication: the client cannot verify the identity of the server

HTTPS is the encrypted variant of HTTP

- HTTPS = HTTP on top of a cryptographic protocol (SSL / TLS)
- HTTPS provides confidentiality, integrity and server authentication
- a necessary ingredient on modern web applications: no excuses!

Background: Symmetric Key Cryptography

In symmetric key cryptography, the encryption and decryption operations use the same key

- encryption and decryption are very efficient
- since the key is a shared secret, it must be established offline (or out of band) before starting communication
- if A and B use a symmetric key to communicate, it is impossible to know which party (A or B) created an encrypted message

Background: Asymmetric Key Cryptography

In asymmetric key cryptography, each party has a key pair including a public key and a private key

- encryption and decryption are less efficient
- encryption with the public key enables decryption with the public key, so there is no need to rely on out-of-band messages
- importantly, encryption with the private key enables decryption with the public key: this supports the implementation of digital signatures

TLS in a Nutshell

Key ingredients:

- asymmetric cryptography: used to establish a symmetric key, called the session key, between the client and the server
- symmetric cryptography: used to ensure the confidentiality and the integrity of the exchanged messages by means of the session key
- certificates: used to provide authentication by binding the public key of the server to its identity. This is done thanks to a signature issued by a trusted certification authority

TLS in a Nutshell

High-level view:

1. The client initiates a handshake with the server by proposing a TLS version and a list of supported cipher suites
2. The server chooses the lower between its highest supported TLS version and the TLS version proposed by the client. It then picks a supported cipher suite from the proposed list and sends its certificate
3. The client confirms the validity of the certificate and retrieves the server's public key from it
4. The client and the server take appropriate actions to generate the session key, taking advantage of the server's public key
5. The session key is used to protect the communication

Session Key Establishment

A classic approach to session key exchange is dubbed RSA:

1. the client generates a random number s
2. the client sends s to the server, encrypted with the server's pub key
3. the server decrypts s using its own private key
4. the client and the server derive the session key k from s

The problem with this approach is the lack of forward secrecy: if the server's private key is leaked, then all the established session keys are compromised as well!

Session Key Establishment

An alternative approach is the Diffie-Hellman key exchange:

1. the client and the server publicly agree on a prime p and a base g
2. the client generates a secret a and sends $g^a \bmod p$ to the server
3. the server generates a secret b and sends $g^b \bmod p$ to the client, signed with its own private key
4. the client verifies the signature using the server's public key
5. the client generates the session key $k = (g^b)^a \bmod p$
6. the server generates the session key $k = (g^a)^b \bmod p$

Contrary to RSA, Diffie-Hellman provides forward secrecy. Why?

Cipher Suites

TLS requires the agreement on cryptographic algorithms for:

1. the asymmetric crypto scheme used to establish the session key
2. the asymmetric crypto scheme used to sign the server's certificate
3. the symmetric crypto scheme used to encrypt the session data
4. the symmetric crypto scheme used to prove the integrity of the session data

All this information is condensed into a ciphersuite, like:

```
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
```

More details on configuration options are available online.

Certificates

A certificate is a cryptographic proof of the ownership of a public key. It includes information about:

1. Serial Number: used to handle certificate revocation
2. Subject: the owner of the public key
3. Public Key: the public key of the subject
4. Issuer: the entity who released the certificate
5. Signature: a signature of the certificate body with the issuer's private key

Browsers come with a list of trusted certification authorities who are entitled to issue valid certificates: their public keys are preloaded.

Certification Authorities

The certificate ecosystem is built on top of a chain of trust.

Definition

A certificate chain is a list of certificates such that:

1. the Issuer of each certificate (except the last one) matches the Subject of the next certificate in the list;
2. each certificate (except the last one) can be verified using the public key contained in the next certificate in the list;
3. the last certificate is a trust anchor: a self-signed certificate that you trust because it was preloaded in the browser.

Types of Certificates

Certificates can have three different validation levels:

1. Domain Validation: bind a public key to a domain name, i.e., they are issued after showing the right to manage a domain name
2. Organization Validation: bind a public key to an organization, i.e., they additionally require a proof of existence as a legal entity
3. Extended Validation: similar to Organization Validation, but with even stricter rules aimed at assessing the legitimacy of the company

Browsers normally use different visual indicators for EV certificates, e.g., a green address bar, but details vary.

Scope of Certificates

Certificates can be issued for:

1. Single domain: valid for a domain name and the `www` sub-domain
2. Multiple domains: valid for an arbitrary list of domain names, set in the Subject Alternative Name field of the certificate
3. Arbitrary sub-domains: valid for a domain name and all its first-level sub-domains, e.g., `*.example.com`

The latter two options can be combined in the same certificate.

Notice that EV certificates cannot make use of wildcards, i.e., all the domains covered by the certificate must be explicitly listed therein.

Scope of Certificates

Why should one care about the scope of certificates?

- if two machines share the same certificate, they also share the same public and private keys!
- once a machine is compromised and its private key is leaked, the attacker can impersonate all the other machines sharing the same certificate and possibly inspect traffic exchanged with them
- certificate reuse increases the attack surface against web apps

Real-world services might have hundreds of sub-domains...

Breaking the Chain of Trust

Who certifies the certification authorities?

- In 2011, DigiNotar issued a fraudulent certificate for google.com and all of its subdomains to Iranian hackers, who used the certificate for man-in-the-middle attacks in Iran
- In 2012, Trustwave issued a fraudulent certificate that enabled an unnamed private company to spy on all SSL-protected connections within its corporate network

Certification authorities can issue valid certificates for all domains!

Certificate Transparency

Certificate Transparency (CT) is a public, append-only log of certificates issued by existing certification authorities

- domain owners can check how many certificates have been issued for a domain name and who are the issuers of the certificates
- Google Chrome originally required CT for EV certificates in 2015 and now extended this to all certificates issued after April 2018
- site operators can force browsers to look for the presence of their certificates in the CT logs by using the `Expect-CT` header

Certificate Transparency

The `Expect-CT` header allows sites to opt in to reporting and/or enforcement of CT requirements

- `max-age`: the number of seconds during which the browser should report and/or enforce CT requirements
- `report-uri` (optional): the URI to which the browser should report failures in CT requirements
- `enforce` (optional): instructs the browser to refuse connections that violate CT requirements

Quick Recap

Okay, let's assume we spent enough time to:

- identify an appropriately up-to-date, robust cipher suite
- get a valid certificate, with the "right" type and scope
- appropriately deploy the `Expect-CT` header

All the HTTPS interactions with your site are bullet-proof. What's next?

Elephant in the room

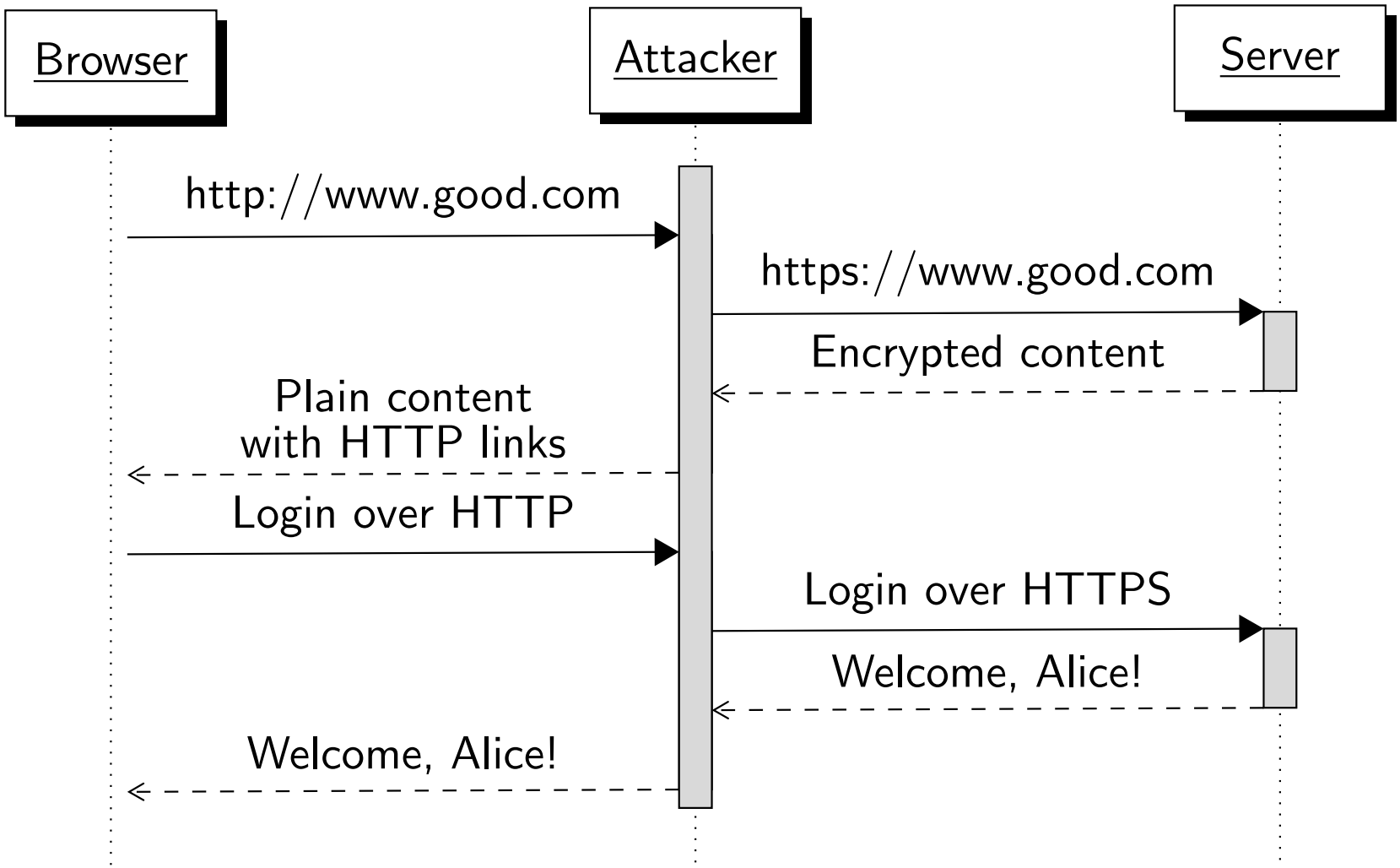
Though HTTPS is in a great shape, HTTP is still around!

SSL Stripping

SSL stripping is a subtle man-in-the-middle attack, exploiting user habits and standard browser behaviour

1. The user navigates the browser to `www.good.com`
2. Since no protocol was specified, the browser contacts the site over HTTP at `http://www.good.com`
3. The attacker intercepts the HTTP request and establishes a secure connection with `https://www.good.com`
4. The attacker rewrites to HTTP all the HTTPS links in the page
5. The attacker serves the modified page to the browser and starts acting as a MITM, so that all the traffic can be read and corrupted

SSL Stripping



HTTP Strict Transport Security (HSTS)

HSTS is a security policy which can be used to prevent SSL stripping

- HSTS instructs the browser that the server only wants to accept HTTPS connections (for a given duration)
- the browser must upgrade all the HTTP requests to the server over HTTPS before they are sent on the network
- if the security of the HTTPS connection cannot be established, e.g., untrusted TLS certificate, the browser must stop the connection

```
Strict-Transport-Security: max-age=31536000
```

HSTS and Cookie Security

The use of HSTS makes the `Secure` attribute redundant for cookies

- ... unless those cookies are shared across sub-domains, by using the `Domain` attribute!
- HSTS can be activated with the `includeSubDomains` option to extend the HTTPS upgrade to sub-domains
- this is useful to enforce cookie confidentiality and integrity
- why don't we just enforce HSTS on each sub-domain?

HSTS Preload List

Problem: before the first HSTS header is sent to the browser and cached, the connection might still be unprotected!

- browsers make use of a preloaded list of known HSTS domains
- domain owners can ask for inclusion in the preload list

Requirements

1. Serve a valid certificate
2. Redirect from HTTP to HTTPS on the same host
3. Serve all sub-domains over HTTPS
4. Serve an HSTS header on the base domain for HTTPS requests, with `max-age` set to at least 1 year and `includeSubDomains`

HSTS Best Practices

A few important points to keep in mind when deploying HSTS:

1. although `foo.com` just implements a redirection to `www.foo.com` after activating HSTS with the `includeSubDomains` option, users might directly access `www.foo.com` and miss HSTS protection
2. to correctly deploy the `includeSubDomains` option, all the entry points of the web application like `www.foo.com` should include a request to a resource from `foo.com` to force HSTS activation
3. `sub.foo.com` is not a sub-domain of `www.foo.com`: activating the `includeSubDomains` option on the latter does not suffice!

CSP for TLS Enforcement

CSP provides facilities to simplify a full transition from HTTP to HTTPS

- `upgrade-insecure-requests`: all the HTTP requests sent from the web page are automatically upgraded to HTTPS before they are sent on the network
- `block-all-mixed-content`: tells the browser that even passive mixed content should not be allowed on the web page, e.g., HTTP requests for images will be blocked

Note that the former directive takes precedence over the latter.