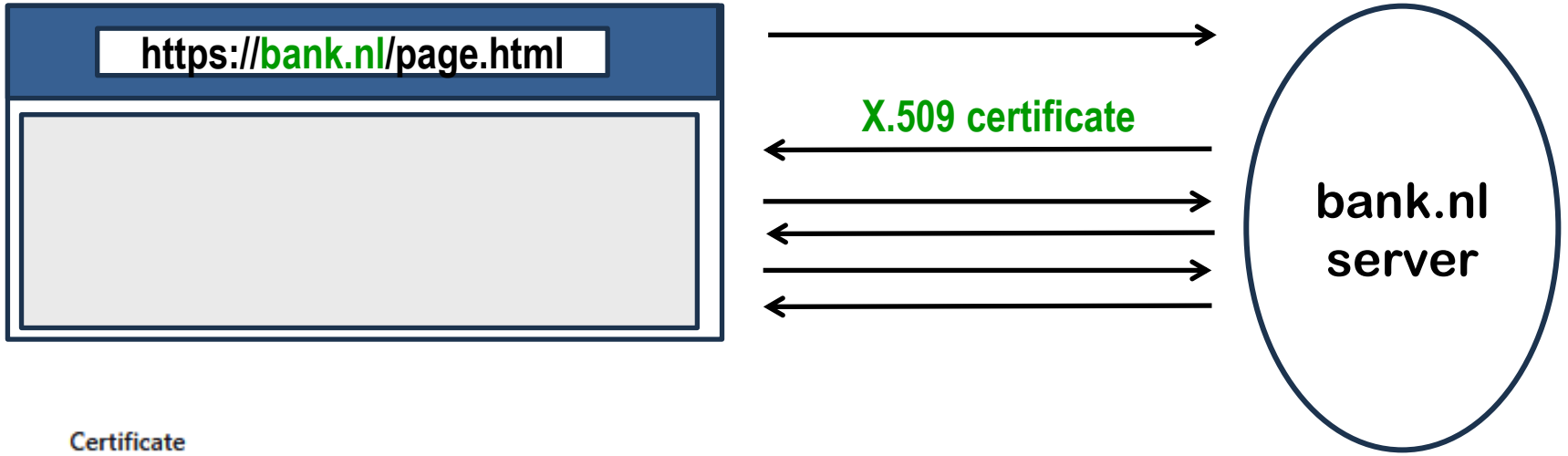
Attacking sessions

Güneş Acar & Erik Poll

Digital Security group
Radboud University Nijmegen

Recap: sessions at network level (TLS / HTTPS)



Certificate

rabobank.nl | Sectigo ECC Extended Validation Secure Server CA | USERTrust ECC Certification Authority

Subject Name

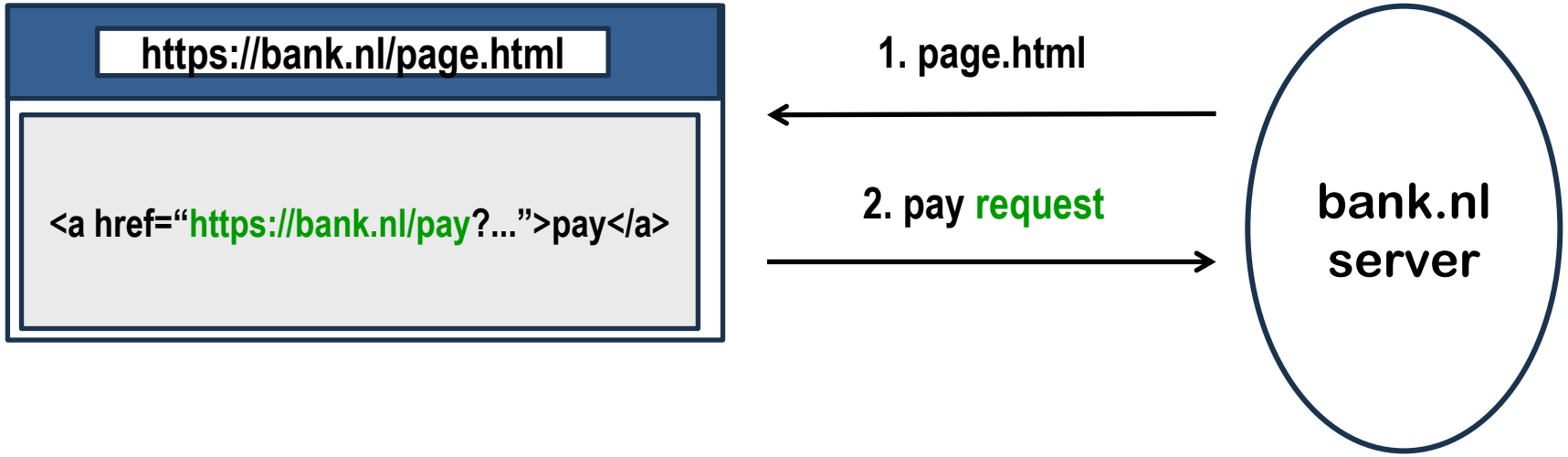
Serial Number	30046259
Inc. Country	NL
Business Category	Private Organization
Country	NL
State/Province	Utrecht
Organization	Cooperatieve Rabobank U.A.
Common Name	rabobank.nl

Issuer Name

Country	GB
State/Province	Greater Manchester
Locality	Salford
Organization	Sectigo Limited
Common Name	Sectigo ECC Extended Validation Secure Server CA

Validity

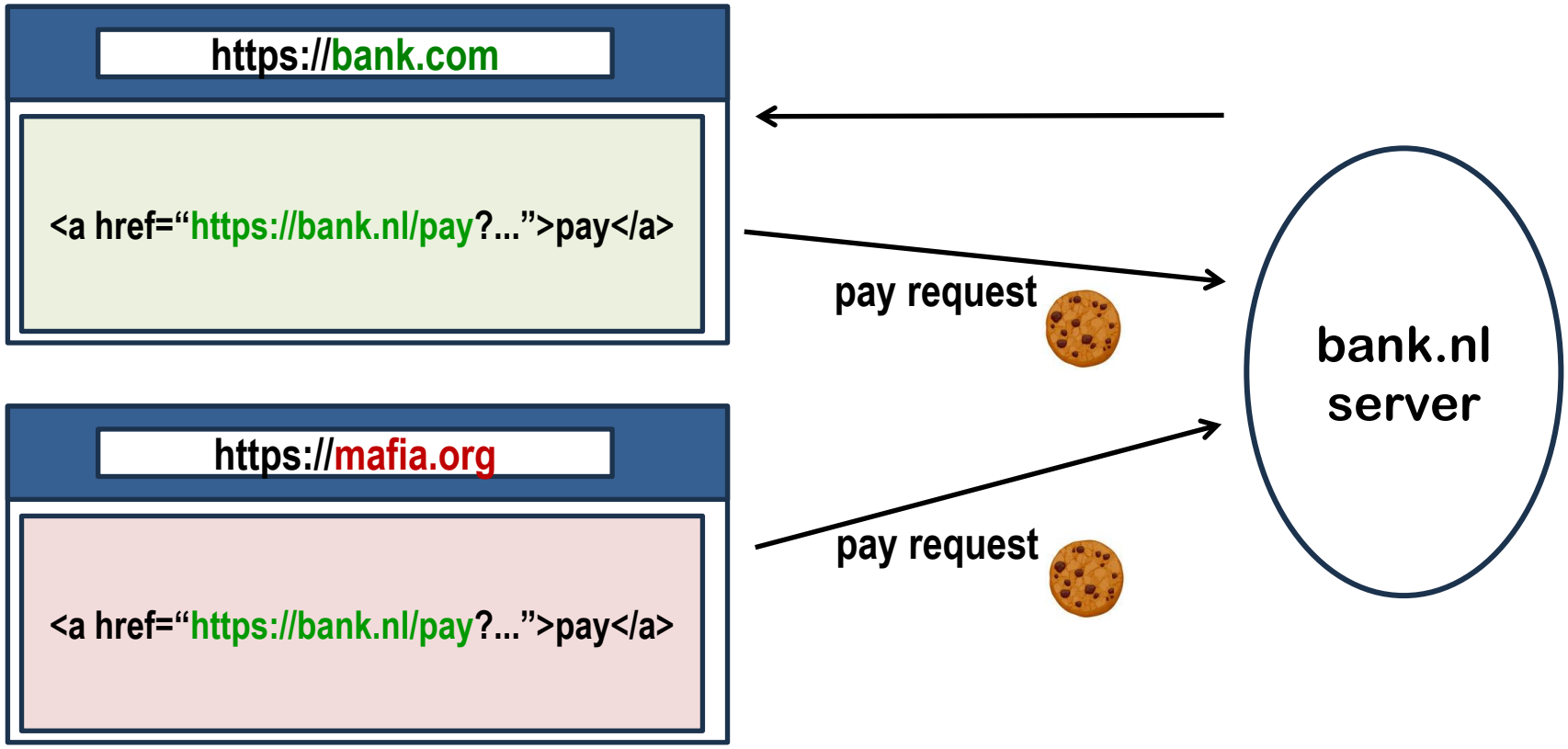
Recap: sessions at application level



Session identifier (aka session token or session id) in HTTP request

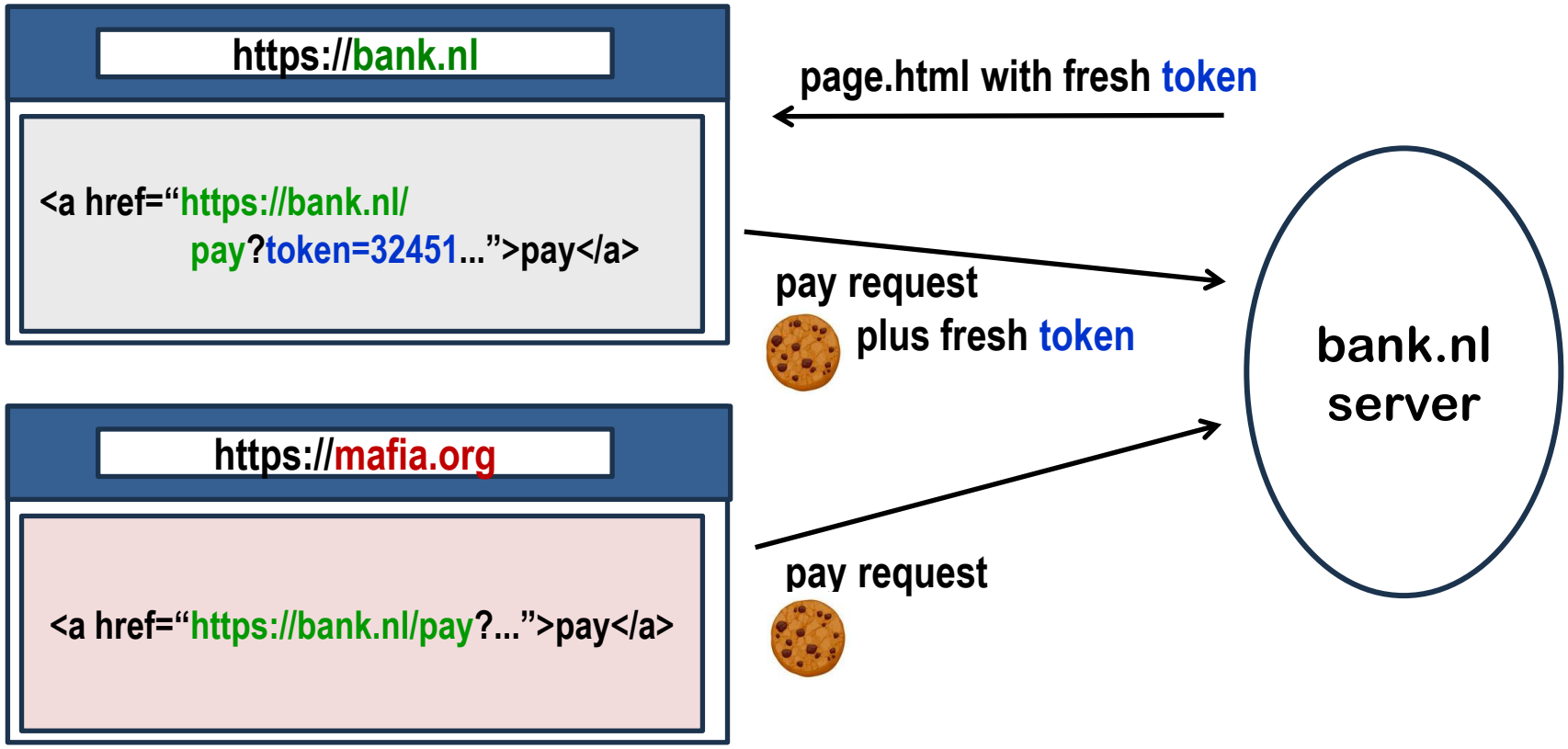
e.g. as URL parameter or cookie in HTTP request

Attack 1) CSRF - the downside of cookies



Browser attaches cookie to cross-domain requests from any site, including cross-site requests from mafia.org to bank.nl

Countermeasure: 1. (anti)CSRF token



Countermeasures against CSRF

1. Anti-CSRF token with an constantly changing value in addition to the constant session id

2. Check `Refer(r)er` or `Origin` headers

   Browser includes these in HTTP requests to indicate where request is made from (eg mafia.com or bank.nl), so bank.nl can check which webpage made the request.
   But... these headers may be absent

   `Origin` is just the domain, `Referer` the domain plus the path

3. Let client re-authenticate before important actions

   Eg. when resetting their password, or making big bank transfer

Countermeasures against CSRF: `SameSite`

4) Set `SameSite` flag for the cookie (since 2017)

- `strict` cookie never attached to cross-site requests
- `lax` cookie only attached to top-level GET requests

  i.e. GET requests that change the address bar to bank.nl

  (so not for loading an iframe from bank.nl on mafia.com)
- `None` cookie is attached to cross-site requests

https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie/SameSite

Countermeasures against CSRF: `Sec-Fetch-Site`

5. Browsers includes

   `Sec-Fetch-Site:<value>`

   in HTTP requests where `<value>` is

   - `none` if user types in the URL or selects a bookmark
   - `cross-site` if user clicks link to `bank.com` on page from say `mafia.com`
   - `same-site` or `same-origin` if user clicks link to `bank.com` on page from `bank.com`
     - `same-site` if scheme (ie https vs http) and domain are the same; `same-origin` if port number is also the same

Note: similar to `Referer` or `Origin`, but without privacy drawbacks

Introduced 2019, available in all browsers since 2023

Check if your browser support this on
https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Sec-Fetch-Site

Attack 2) *Stealing cookies with scripts (XSS)*

If **attacker can inject malicious JavaScript into pages of bank.nl, this code can inspect cookies, URL parameters or any other session information in webpages and send them anywhere on the internet, including to https://mafia.com**

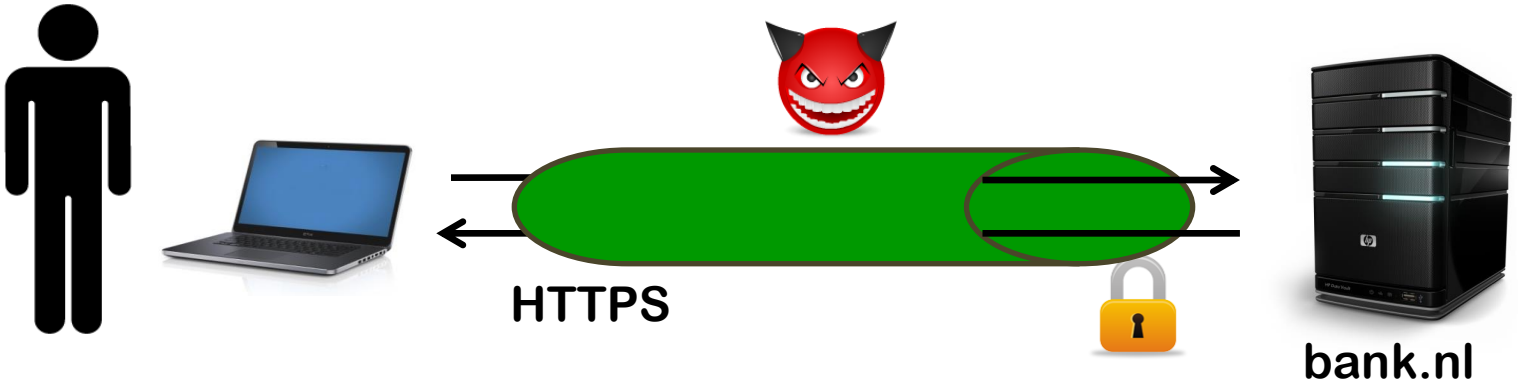
- **Such an attack, called Cross Site Scripting (XSS) attacks will be discussed next week**

- **Solution: cookie has be declared as `HttpOnly`**
  - **This means the cookie is only used for sending along with HTTP requests & not accessible to scripts in the webpage**

Attack 3) *Stealing cookies as MitM*

HTTPS prevents eavesdropper from observing cookies

(and other session info)



But: active MitM attacker can so more…

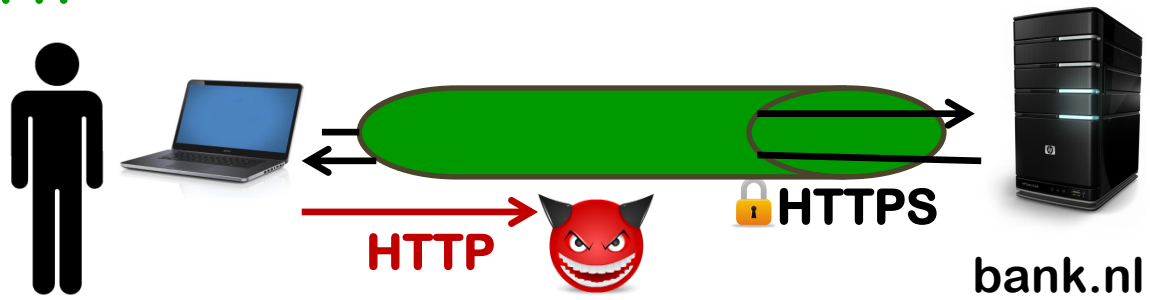
How can you do a MitM attack?

1. **attacker on (wired/wireless) network**
2. **ISP**
3. **fake website** eg. bank.ni

Stealing cookies as MitM

1. User logs on to https://bank.nl
2. Server sets session ID for bank.nl in cookie
   - which is encrypted in HTTPS-traffic, so attacker cannot steal it
3. User does their banking
   ....
4. User asks for unencrypted HTTP link (eg http://nu.nl)
5. MitM attacker replies with a redirect to http://bank.nl
6. Browser follows redirect and sends the bank's cookie over HTTP
7. Bingo! Attacker has the cookie

Solution: set `secure` flag for cookie which disallows browser to *ever* send it over HTTP



Recap: cookie flags

The three cookie flags

1. `secure` only ever sent cookie over HTTPS, never over HTTP

   Encrypting the cookie itself, when it is sent over HTTP, is pointless.

   Why?

   Attackers can simply replay a stolen encrypted cookie!

2. `HTTPonly` inaccessible to scripts

   Confusing name: `HTTP(S)Only` would be better

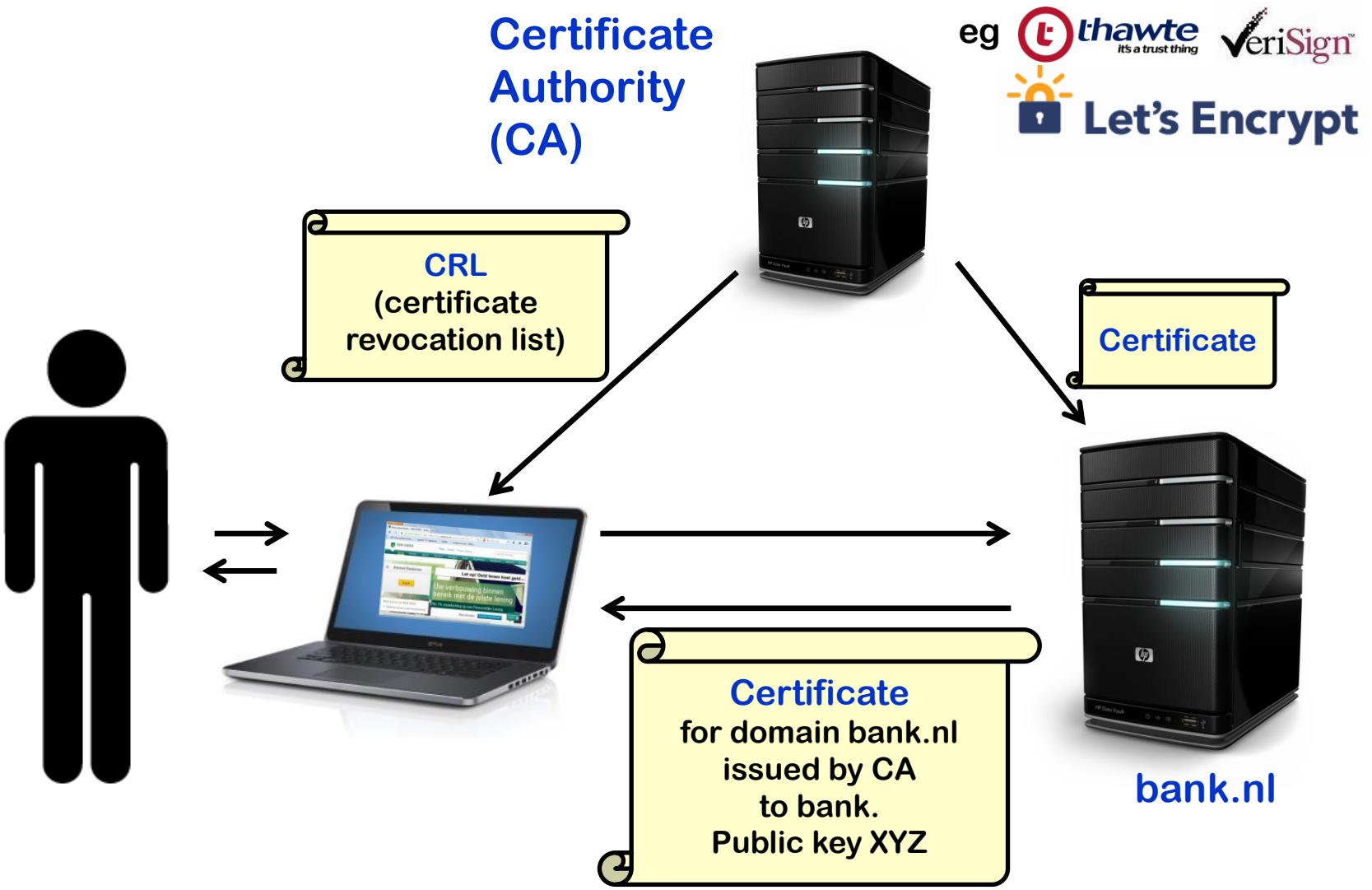
3. `SameSite` not added to links from different sites

protect against three different attacker models

1. protects against eavesdropping and MitM
2. protects against XSS: client-side scripts injected into the 'real' website (discussed in later lecture)
3. protects against malicious sites that link to a benign site (CSRF)

Attacking HTTPS

What are we trusting? And for what?



TCB

- **TCB = Trusted Computing Base**

- **TCB *of a security control or security guarantee* is everything (people, software, computers, …) that we have to trust for that control or guarantee**

- **There will be different TCBs for different controls/guarantees**

Things that go wrong ...

Certificate Authority DigiNotar
was hacked in 2011

Fake certificates for google.com
were issued, presumably for use in Iran

DigiNotar provided *all* the
certificates for the NL government...

DigiNotar
Internet Trust Services

How was this problem detected?
Chrome browser checked for suspicious certificates for google.com

Darknet Diaries has a great podcast episode on this:
https://darknetdiaries.com/episode/3/

How trustworthy are Certificate Authorities (CAs)?

Google Chrome to distrust Chunghwa Telecom, Netlock certificates in August

By **Bill Toulas** June 2, 2025 01:36 PM 0

https://www.bleepingcomputer.com/news/security/google-chrome-to-distrust-chunghwa-telecom-netlock-certificates-in-august

Mysterious company with government ties plays key internet role

TrustCor Systems vouches for the legitimacy of websites. But its physical address is a UPS Store in Toronto.

By Joseph Menn

Updated November 8, 2022 at 5:37 p.m. EST | Published November 8, 2022

Google's Chrome, Apple's Safari, nonprofit Firefox and others allow the company, TrustCor Systems, to act as what's known as a root certificate authority, a powerful spot in the internet's infrastructure that guarantees websites are not fake, guiding users to them seamlessly.

The company's Panamanian registration records show that it has the identical slate of officers, agents and partners as a spyware maker identified this year as an affiliate of Arizona-based Packet Forensics, which public contracting records and company documents show has sold communication interception services to U.S. government agencies for more than a decade.

https://www.washingtonpost.com/technology/2022/11/08/trustcor-internet-addresses-government-connections

Certificate Transparency (CT)

Solution to detect *mis-issued* certificates using a CT log

issued by corrupt insider, by a hacked CA, or by mistake

CT log: public append-only list of *all* issued certificates by *all* CAs

See https://crt.sh

Two ways to use it

1. Organisation can spot rogue certificates issued in their name

   Someone at Radboud could/should periodically check for rogue certificates issued for ru.nl

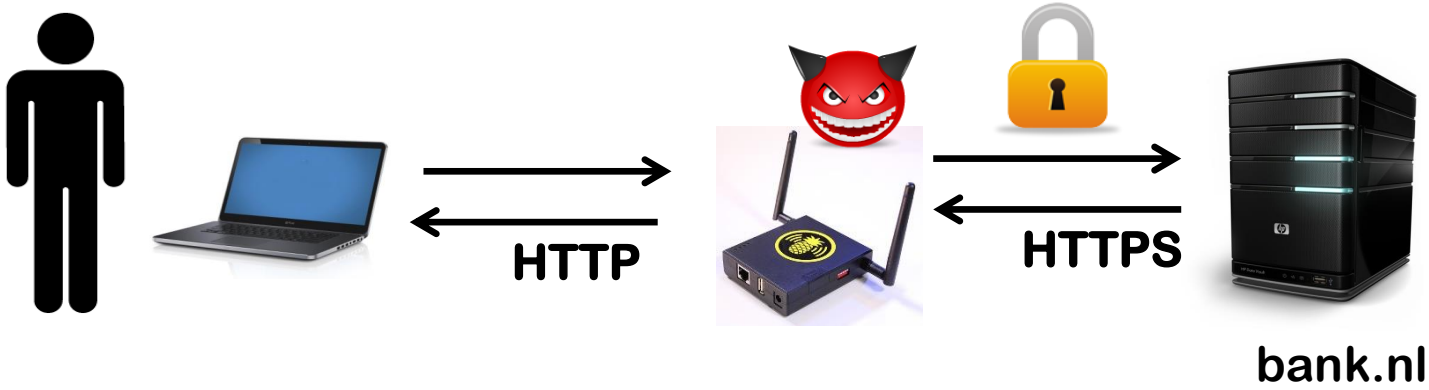
2. Browser can check if certificate is in the log

   Certificate submitted for inclusion in CT log gets signed timestamp to prove it has been included. So browser can check this timestamp.

   Different browsers use different policies to accept or warn about certificates that miss such a proof

Simple SSL stripping : HTTP + HTTPS

The idea: the attacker forces the browser to fall back to HTTP and hopes the user won't notice the missing s

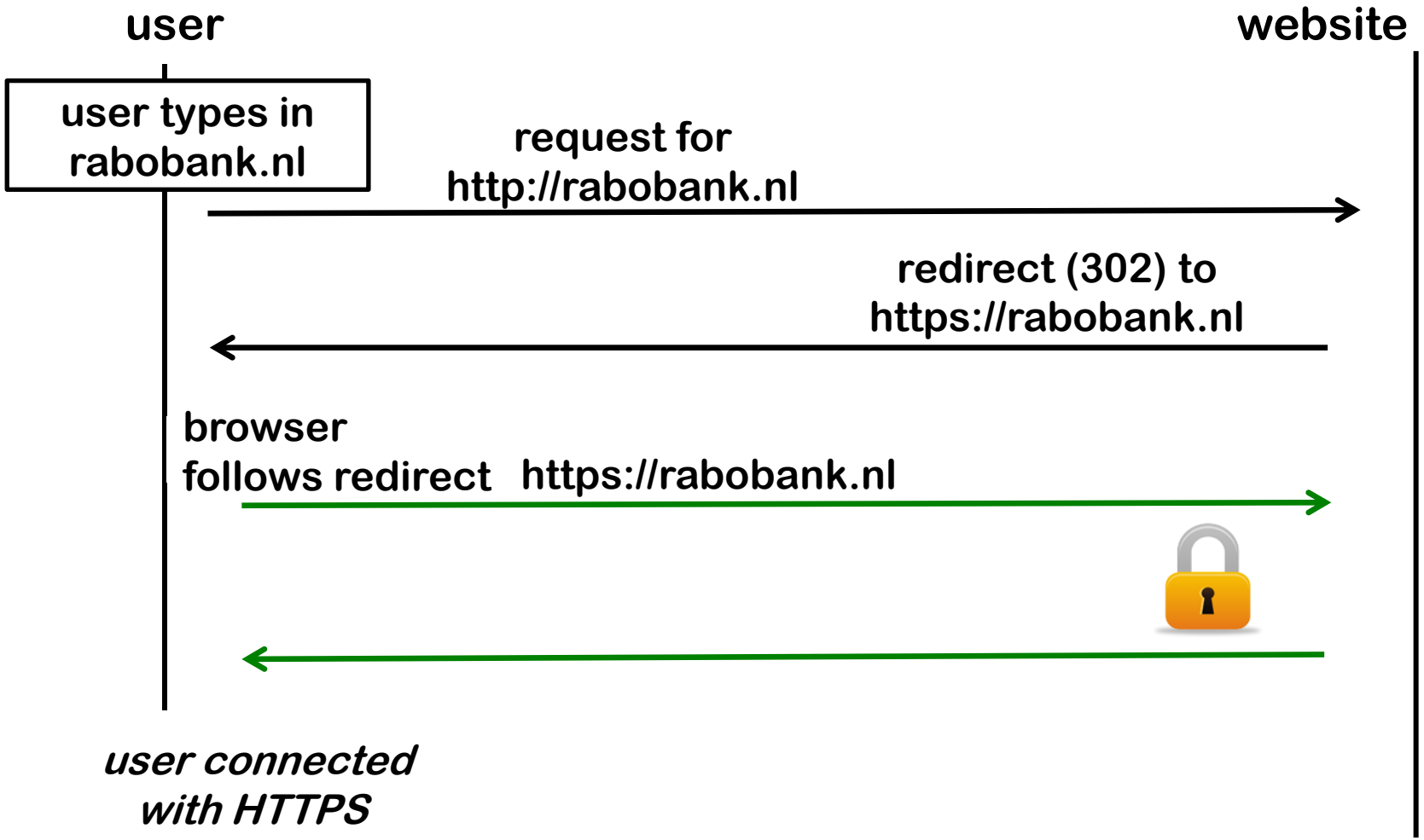


When can the attacker do this? If the user

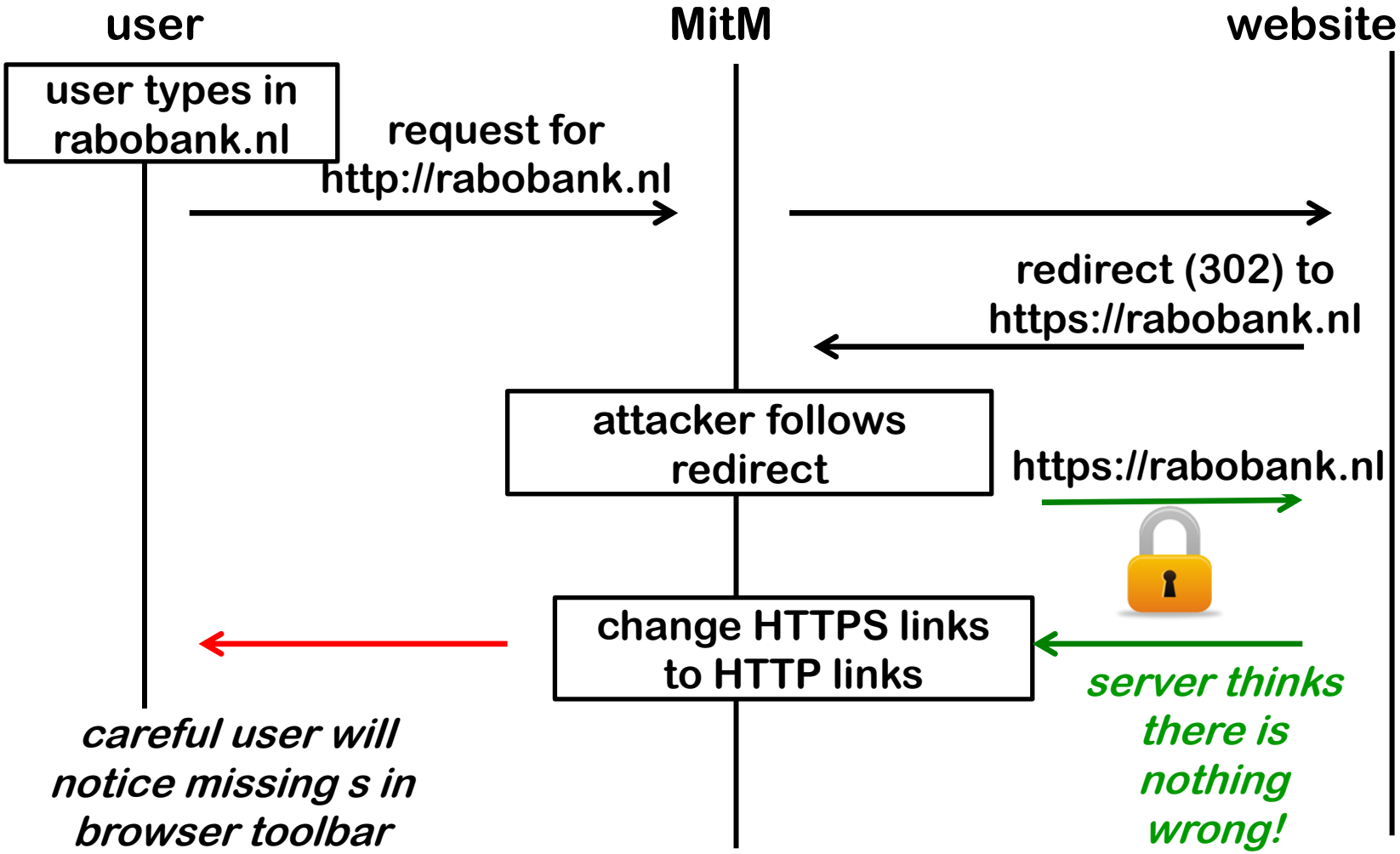
a) types in rabobank.nl, without https in front of it

b) begins a HTTPS session by clicking on a link in a webpage that was retrieved with HTTP

Normal start of HTTPS session via HTTP request

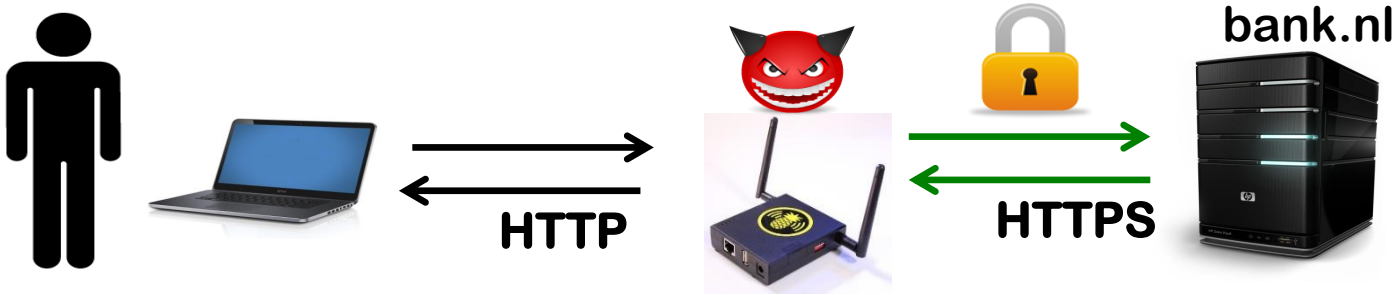


MitM attack on such a session



SSL stripping

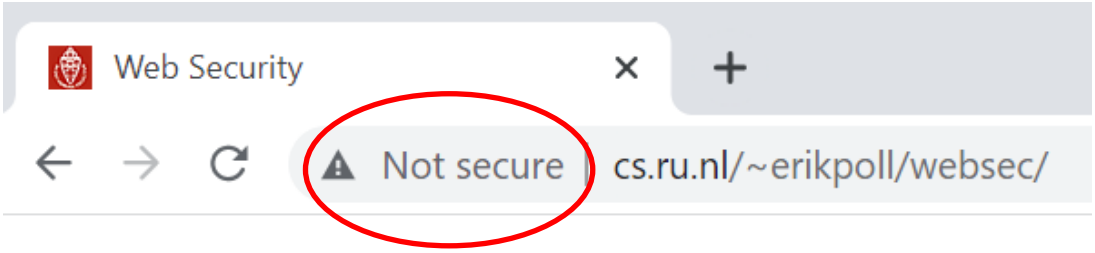
- The MitM attacker
  - strips S from HTTPS in links in traffic from server to user
  - puts this S back in traffic from the user to the server
- The result



- The attacker can now intercept a username and password that the user sends
- After intercepting this information, the attacker could stop the MitM attack
  - and the user can then no longer see anything wrong!
- Attacker could also make arbitrary alterations to the web page

Spotting this attack?

Modern browsers will warn about “insecure website”



In older browsers, only very careful users would spot this attack by noticing that the URL misses an s in https

Countermeasures to SSL stripping

- HTTPS Everywhere browser plugin
  - ie. simply never use HTTP

    https://www.eff.org/deeplinks/2021/09/https-actually-everywhere

- HSTS (HTTP Strict Transport Security)

HTTP Strict Transport Security (HSTS) [RFC6797]

Protection against SSL stripping

1. website (e.g. `bank.nl`) tells browser that it only ever wants to use HTTPS, in HTTP response header

   ```
   Strict-Transport-Security: max-age=15768000;
   includeSubDomain
   ```

2. the browser remembers this, and turn future http requests for that domain into https requests

   Eg browser will turn `http://bank.nl/rekening/...`

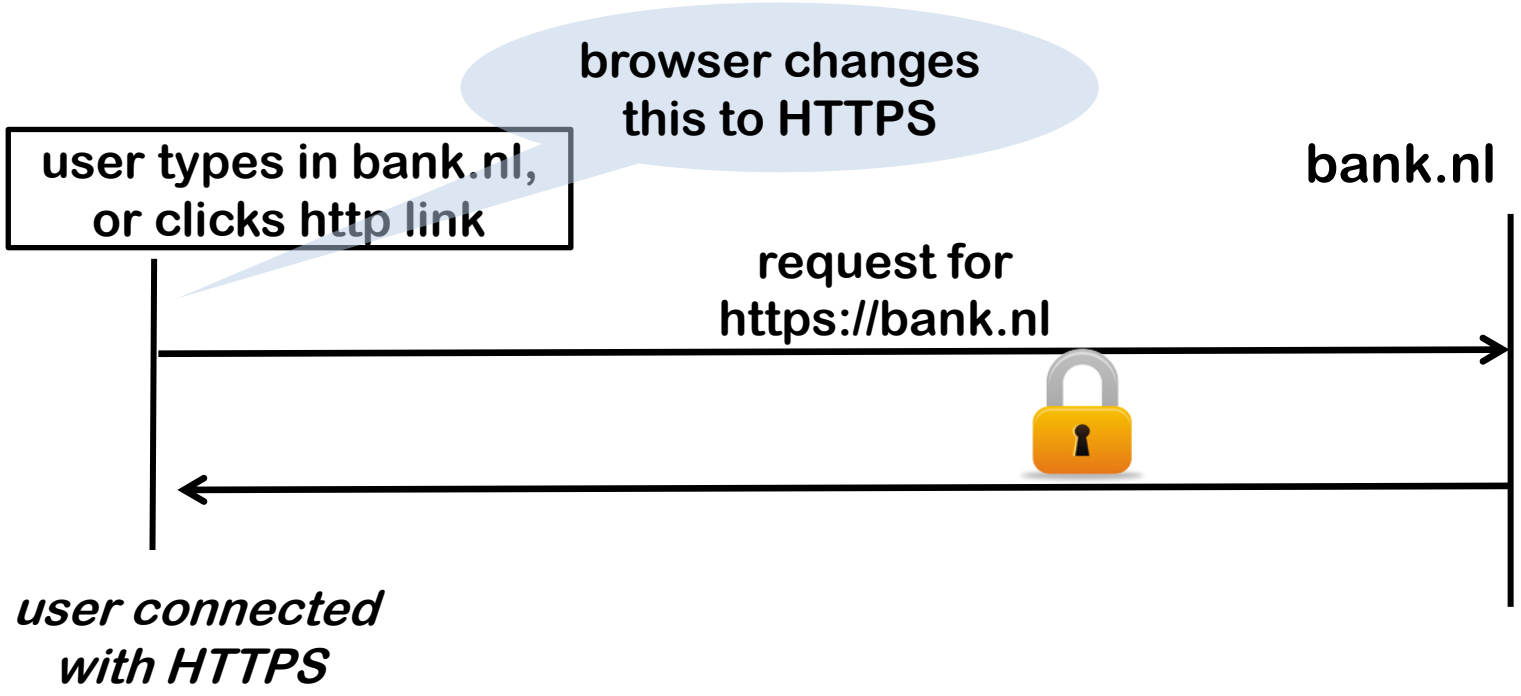
   into `https://bank.nl/rekening/...`

HSTS is now supported by all browsers.

HSTS

On very first visit to bank.nl, **the browser stores some information, recording that bank.nl wants to talk HTTPS only.**

For subsequents visits



Checking for HSTS usage

- **In browser**
  - In Firefox:

    type `about:support` in the address bar. In the Application Basics section, you will see Profile Folder. Click Open Folder, an look for file `SiteSecurityServiceState.txt`

  - In Chrome:

    type `chrome://net-internals/#hsts` in address bar

- **In HTTP traffic:**

  look for HSTS field in HTTP header, of the form

  ```
  Strict-Transport-Security: max-age=15552000; preload
  ```

  On Linux, with `curl -si "https://www.ru.nl" | grep Strict`

Remaining problem with HSTS

- Remaining risk with *very first* request to a site
  - a MitM attacker could SSL strip that first request, and remove the HSTS header in the HTTP response

- Solution: HSTS preload list included in browser that specifies HSTS for some sites, so even first request cannot be with HTTP

  Check https://hstspreload.org/

- New privacy risk with HSTS: HSTS info stored in your browser can reveal which sites have been visited…

  even if you do this in private browsing mode?

Recap

- HTTPS protection fails if attacker can obtain mis-issued certificate
  - CT (Certificate Transparency) can help to detect this

- Active MitM attacker could SSL strip
  - But modern browsers given prominent warnings
  - HTTPS-only and HSTS prevent this
    - Without being on HSTS preload list an HSTS website could still be SSL stripped on a very first visit.
  - Use of HSTS mandatory for Dutch government websites since July 1st, 2023 (Wet digitale overheid)