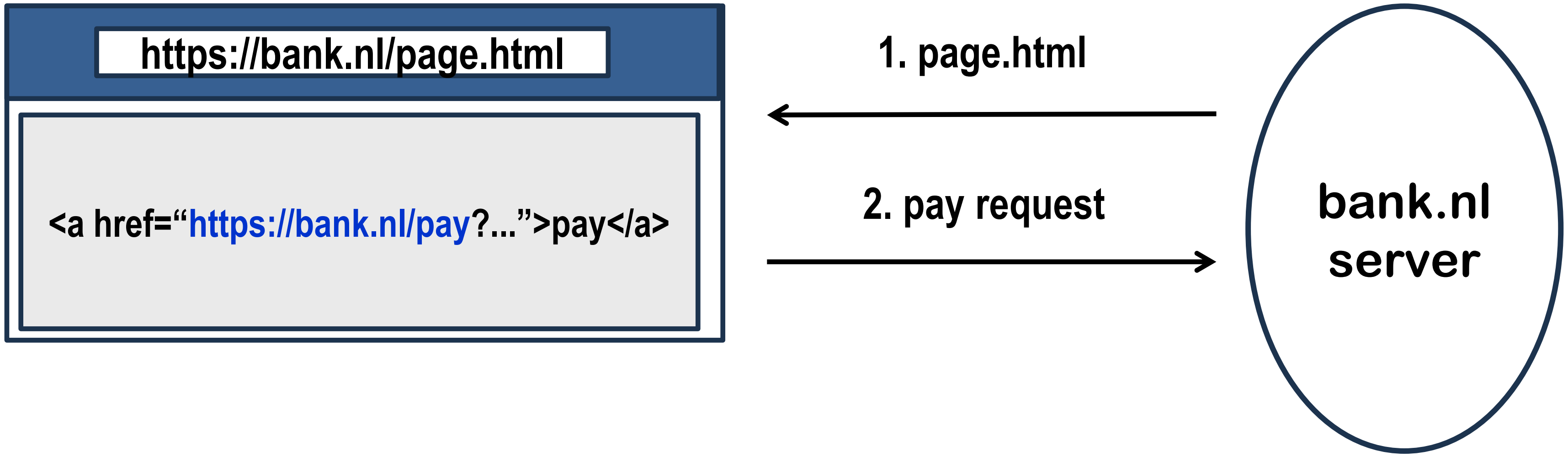
Attacking sessions (continued), at application level & at HTTPS/TLS level

Güne&scedil; Acar & Erik Poll

Digital Security group

Radboud University Nijmegen

Recap: sessions at application level


https://bank.nl/page.html
<a href=“https://bank.nl/pay?...”>pay</a>
1. page.html
2. pay request
bank.nl server

Pay request needs to include session information aka session token(s)

Two ways to do this:

1. Bank provides page.html that includes this info

   as URL parameter (eg below) or in body of HTTP request

   https://bank.nl/pay?sessionID=XYZ123&amount=1.00&dest=12.34.56

2. Bank sets a cookie and browser automatically adds this info

NB: identification vs authentication

Session information serves two distinct purposes:

1. identification

    who is the web server talking to

2. authentication

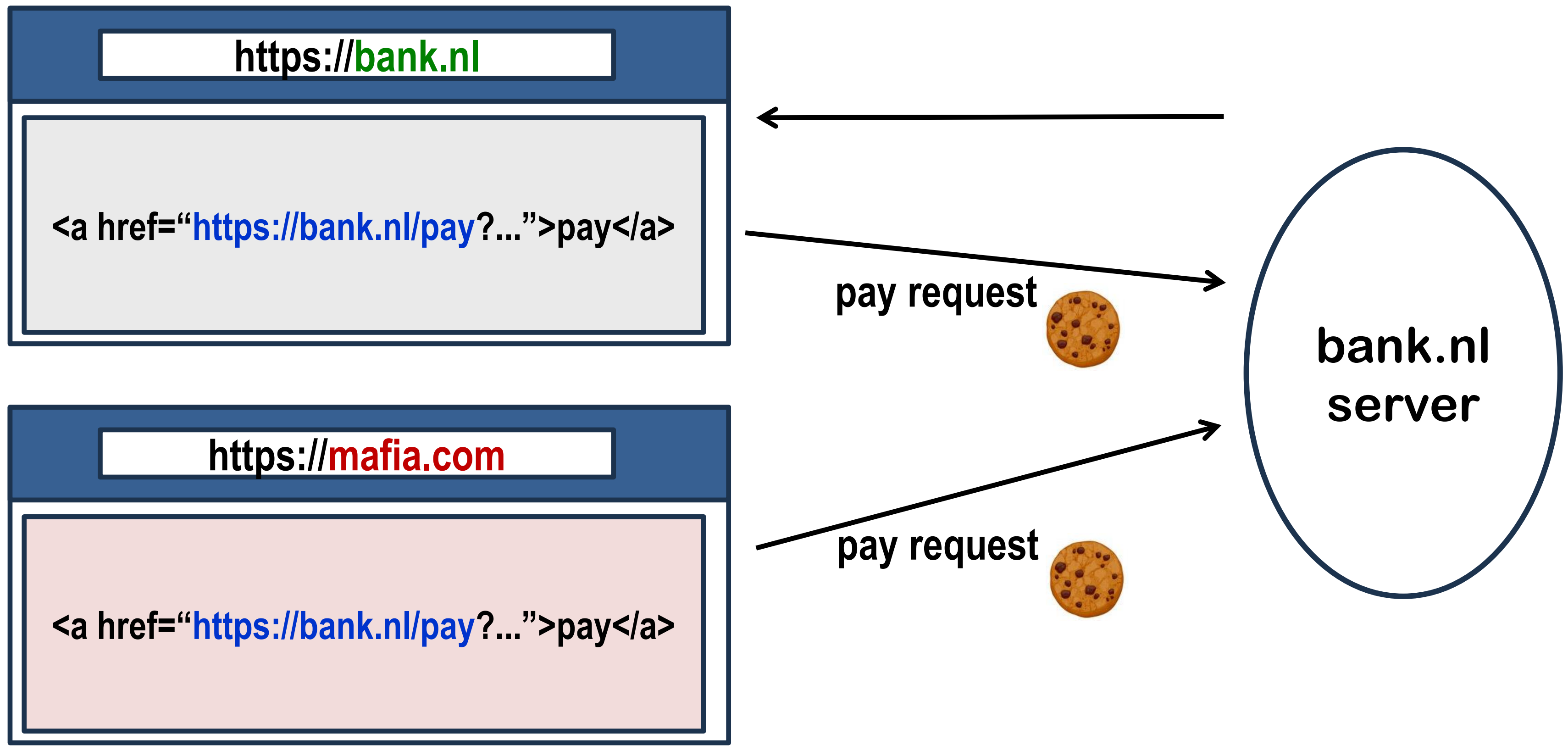
    verifying that this person is who they claim to be

Could Brightspace use your student-nr for this?

For 1, but not for 2

Different pieces of information can be used for 1 and for 2

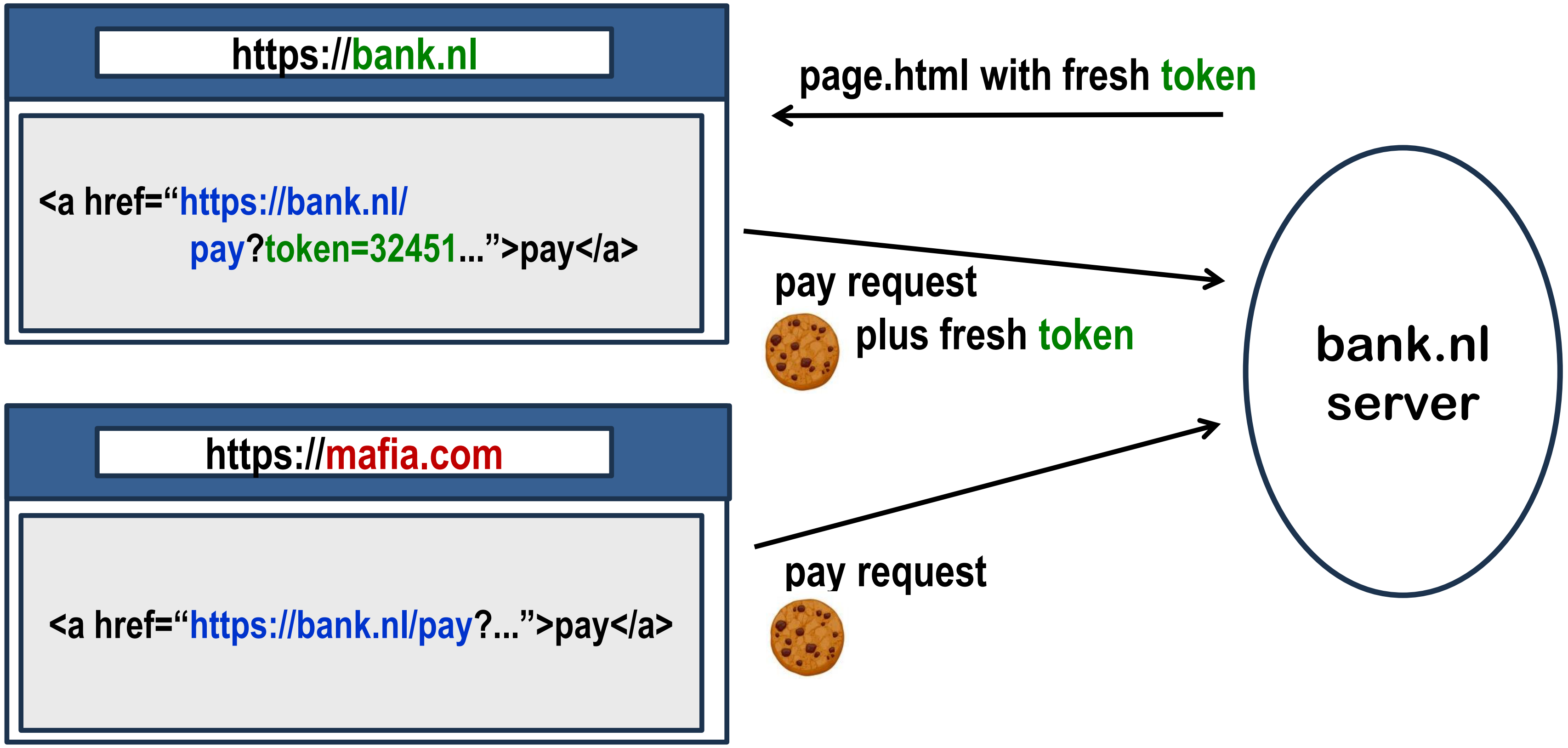
Attack 1) CSRF - the downside of cookies



https://bank.nl
<a href="https://bank.nl/pay?...">pay</a>
https://mafia.com
<a href="https://bank.nl/pay?...">pay</a>
pay request
pay request
bank.nl server

Browser attaches cookie to cross-domain requests from any site

Countermeasure: (anti)CSRF token


https://bank.nl
page.html with fresh token
<a href=“https://bank.nl/pay?token=32451...”>pay</a>
pay request
plus fresh token
bank.nl server
https://mafia.com
<a href=“https://bank.nl/pay?...”>pay</a>
pay request

Other countermeasures against CSRF

- Let client re-authenticate before important actions
  - eg. when resetting their password, or making big bank transfer
- 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 which change the address bar to bank.nl
    (so not for loading an iframe from bank.nl on mafia.com)

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

  `Origin` is just the domain, `Referer` the domain plus the path
- Let browser add `Sec-Fetch-Site` header to distinguish cross site requests (since 2019)

Refer(r)er and Origin

These headers may be absent ☹ :

- Browser can be configured not to include these header, for privacy reasons
- Website can specify Referrer Policy to tell browser not to include them under certain conditions (eg when making HTTP request from HTTPS context, only for requestions within the same site, ...)

https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Origin
https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Referer
https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Referrer-Policy

`Sec-Fetch-Site`

Browser includes `Sec-Fetch-Site:`<value> in HTTP requests

where <value> is

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

Similar to `Referer` or `Origin`, but without privacy drawbacks

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

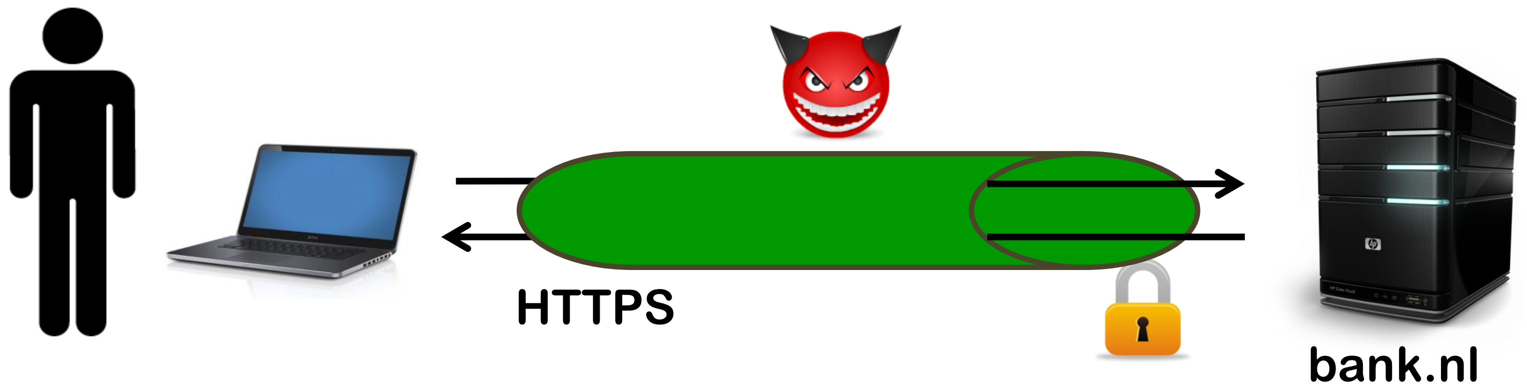
Attack 2) *Stealing cookies with scripts (XSS)*

- If attacker can malicious JavaScript into pages of bank.nl website, this code can inspect the cookies and send them anywhere on the internet, including to https://mafia.com
- Such an attack, called Cross Site Scripting (XSS) attacks will be discussed in next weeks

- 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?

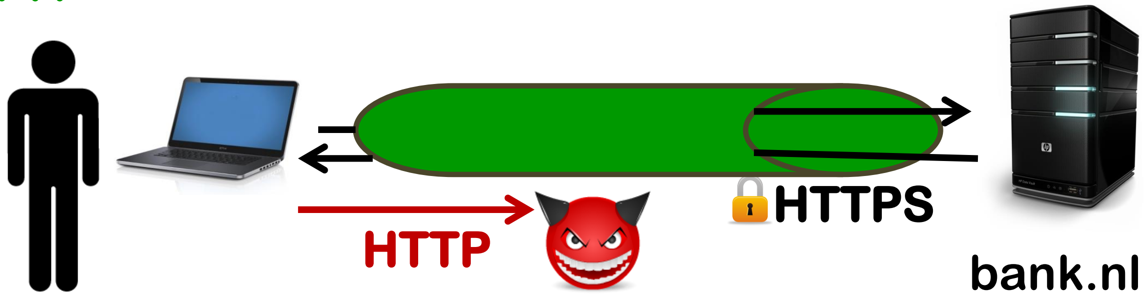
1. attacker on (wired/wireless) network
2. ISP
3. fake website eg. bank.ni

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

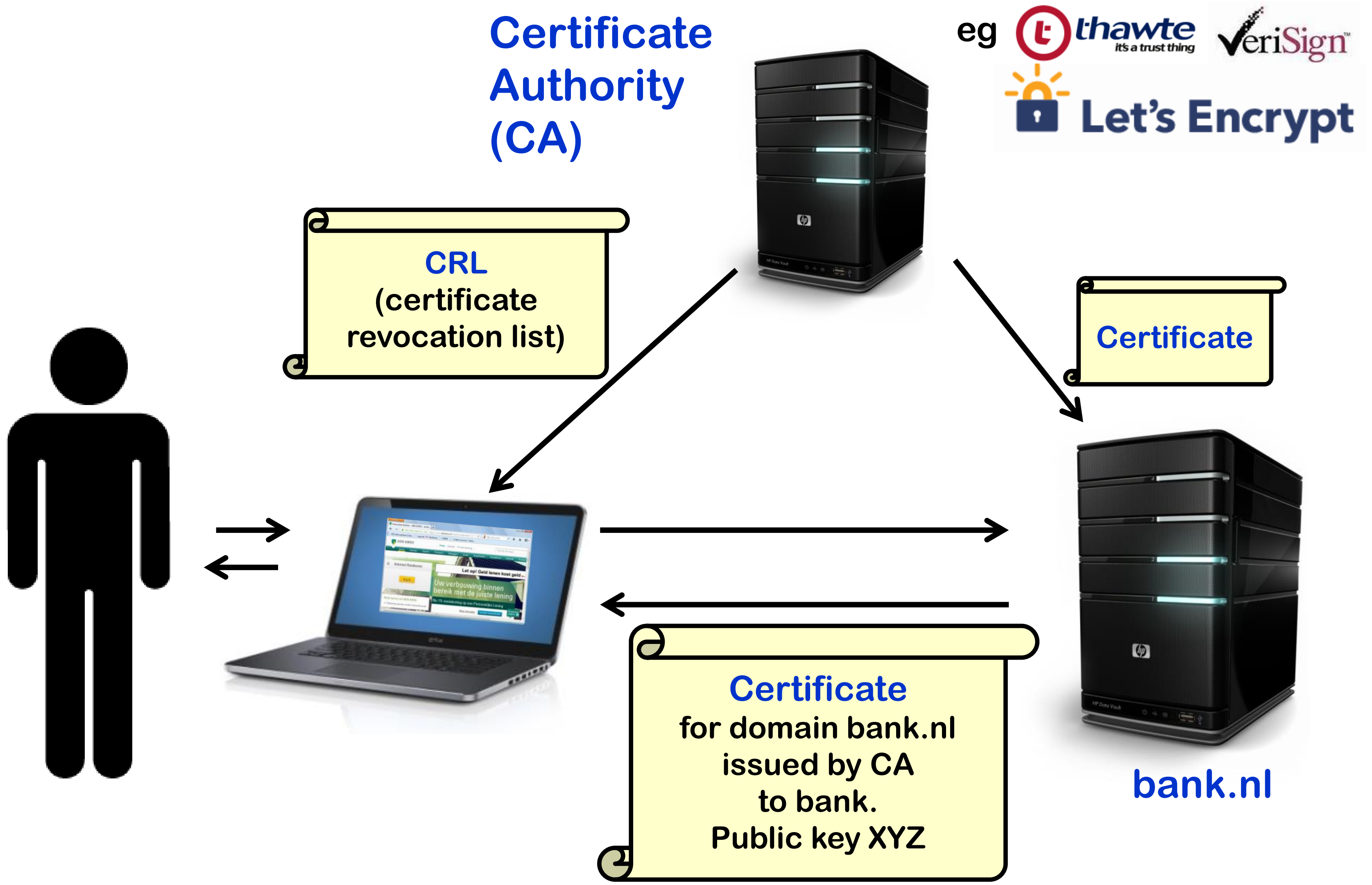
3. `SameSite` not added to links from different sites

protect against three different attacker models

1. protects against eavesdropping and MitM
2. protects against 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?


Certificate Authority (CA)
eg
thawte
it's a trust thing
VeriSign
Let's Encrypt
CRL (certificate revocation list)
Certificate
Certificate for domain bank.nl issued by CA to bank. Public key XYZ
bank.nl

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

DV, OV and EV SSL certificates

Certicate Authority (CA) can validate *who* is requesting a certificate for a domain in different ways:

- DV (Domain Validation) certificates
  - Email check to validate that this is the owner of that domain, using whois information (eg via https://www.sidn.nl/whois)
  - Free via Let'sEncypt since 2016
- OV (Organisation Validation) certificates
  - Additional check on identity & existence of the organisation eg against Chamber of Commerce records
- EV (Extended Validation) certificates
  - More rigorous check on identity of the organisation
  - How much extra security EV gives over OV brings is debatable...

Certificates can be wild-card certificates,

eg for `*.ru.nl` instead of `www.ru.nl`

Things that go wrong ...

Certificate Authority DigiNotar was hacked in 2011.

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

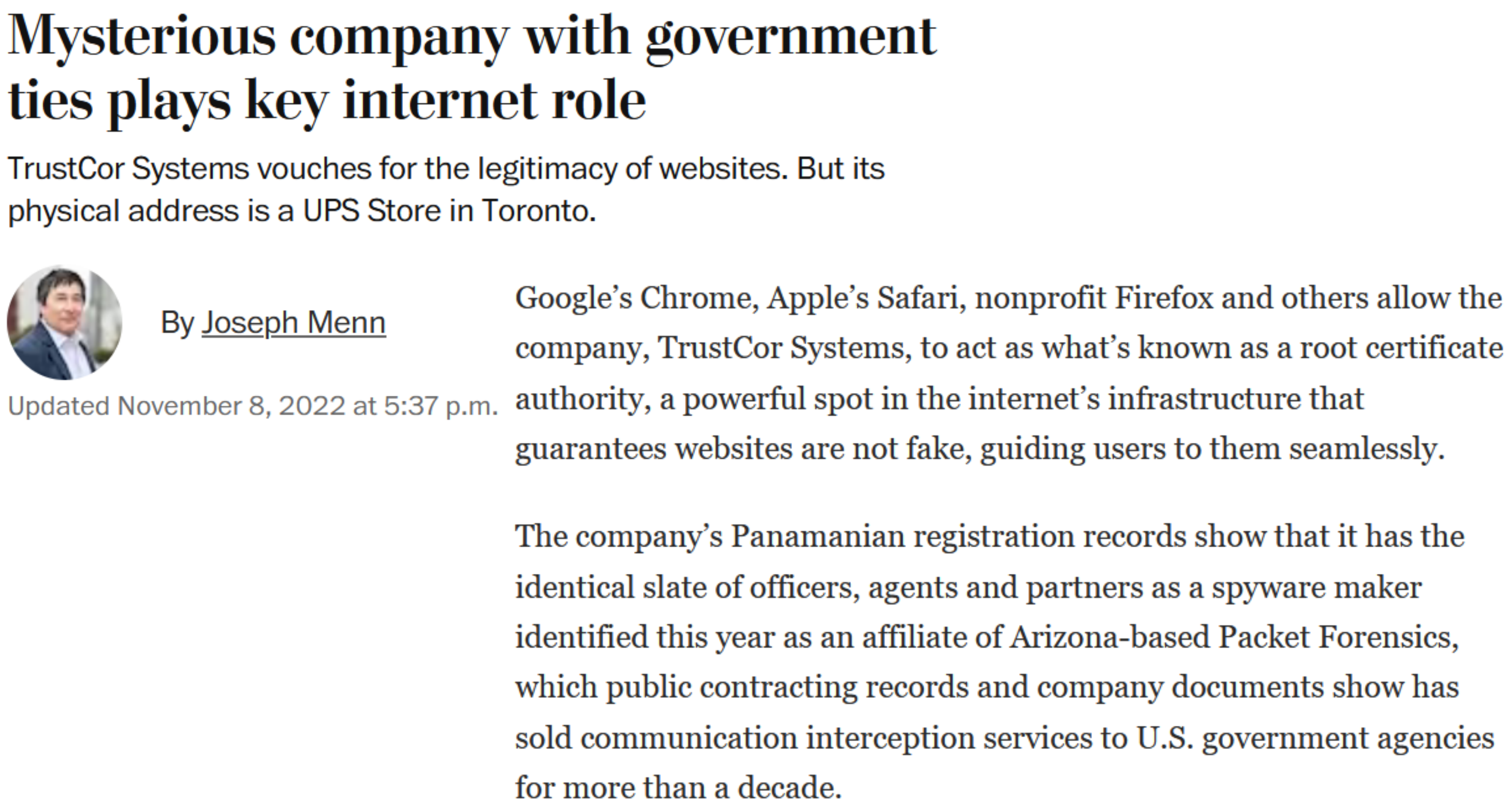
DigiNotar provided *all* the certificates for the NL government...

zeker weten met wie u via internet afspraken maakt
www.diginotar.nl
DigiNotar
Uw elektronische handtekening

Problem detected because the Chrome browser checks for suspicious certificates for google.com

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

Things that go wrong



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.

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

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

- CT is public append-only ledger of *all* issued certificates by *all* CAs
  - See https://crt.sh

- Organisations can spot if a rogue certificate is issued in their name

  i.e. someone at Radboud could/should periodically check there are no rogue certificates issued for ru.nl

- Certificate submitted for inclusion in CT log get a signed timestamp to prove it has been included. So browser can check this. CT log
  - Different browsers use different policies to accept or warn about certificates that miss such a proof

Mixing http & https

A web page can mix http & https content, but this is a bad idea!

- ***Why would you never want to have an frame loaded via http inside a webpage loaded via https?***

Web browsers nowadays warn about or block mixed http/https content.

Demo: check out how this works in your browser, by visiting

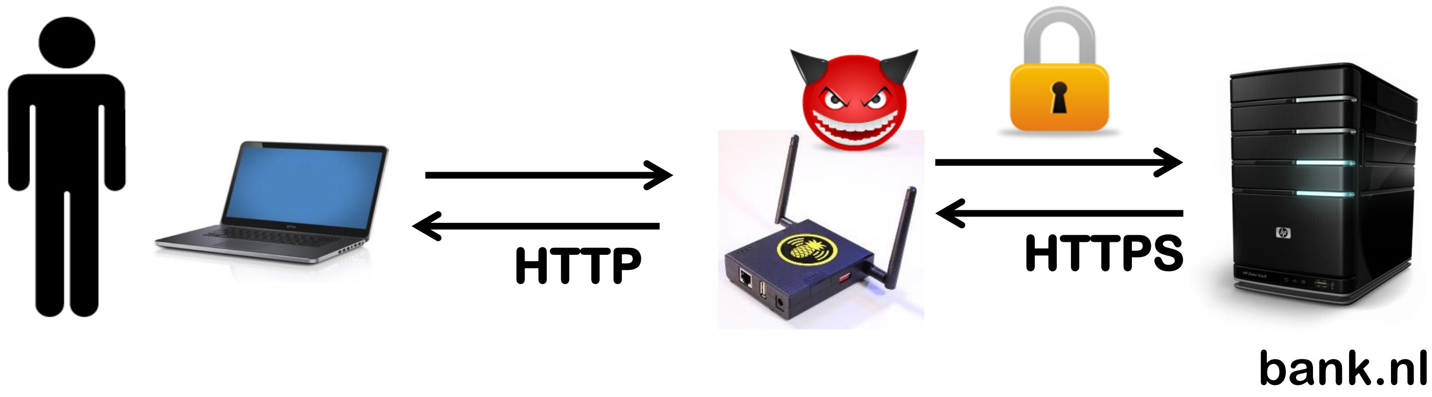
http://www.cs.ru.nl/~erikpoll/websec/demo/mixed_content.html

https://www.cs.ru.nl/~erikpoll/websec/demo/mixed_content.html

This demo no longer works in Firefox, but it does in Chrome

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

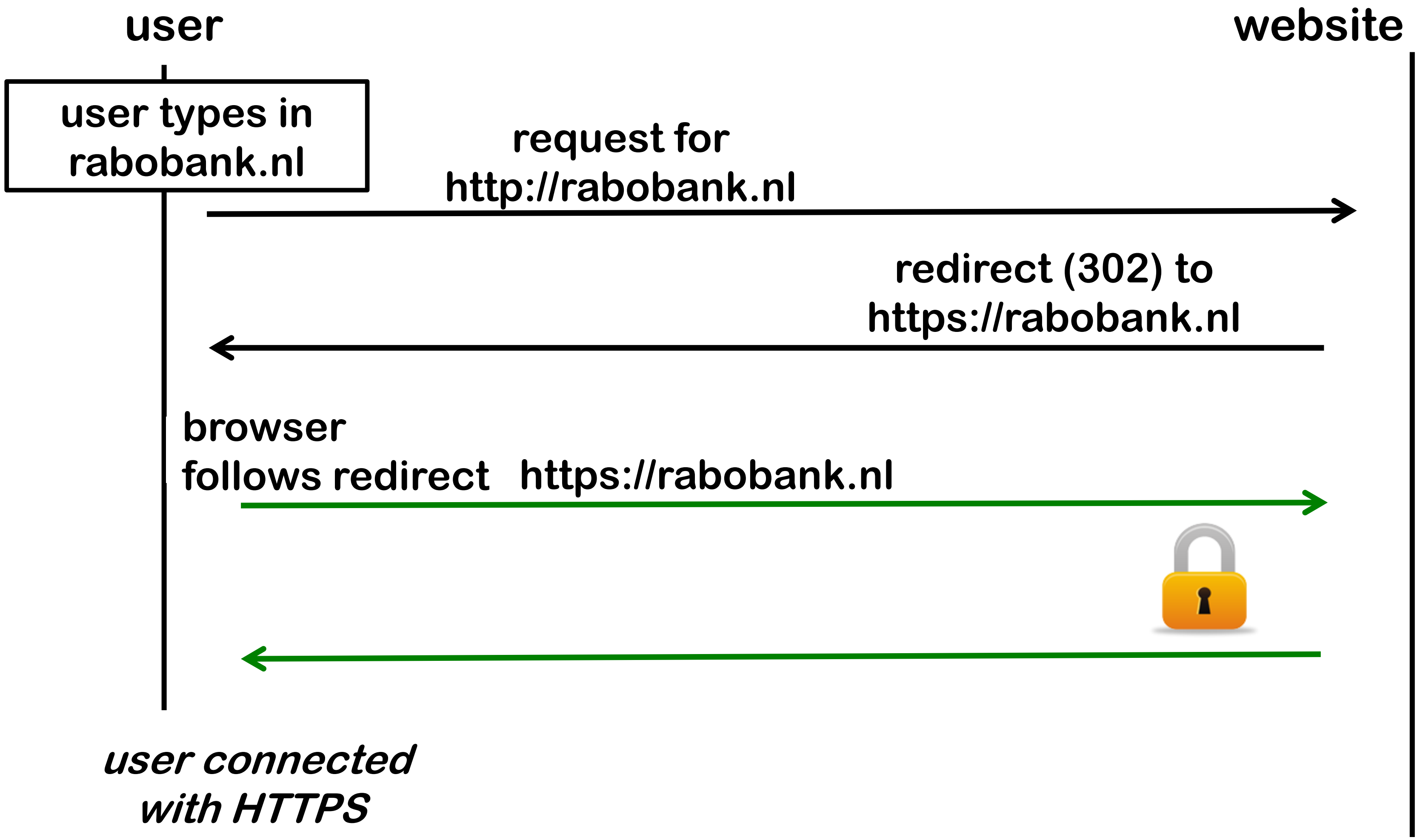


When can the attacker do this? If the user

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

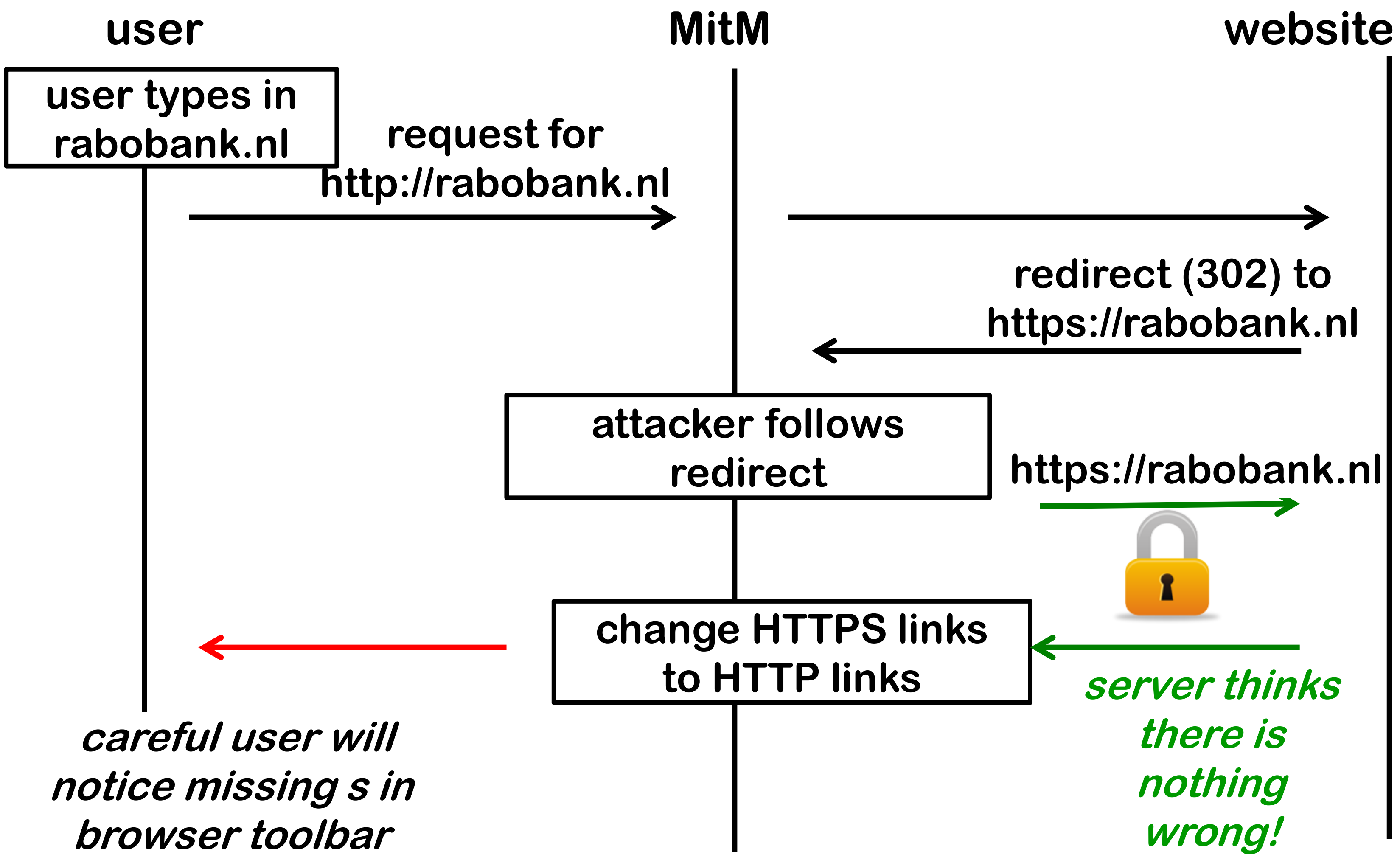
b) begins a HTTPS session by clicking on a link in a webpage that was retrieved with HTTP

Normal start of HTTPS session via HTTP request



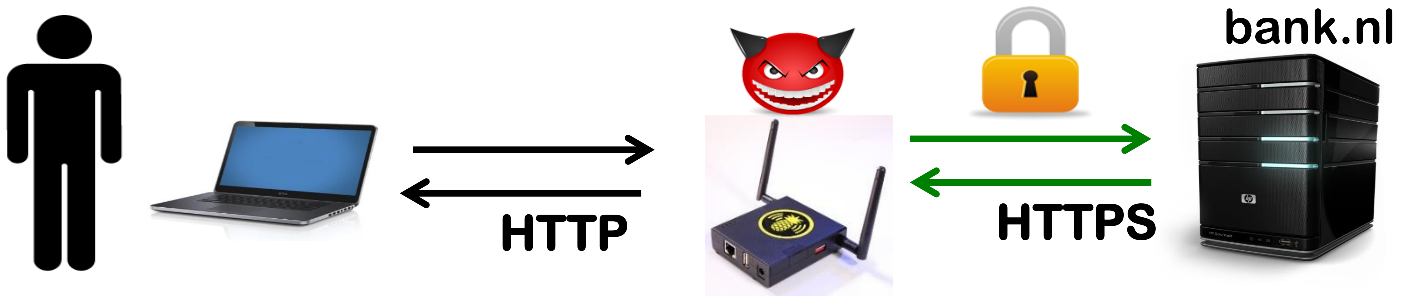
user
website
user types in rabobank.nl
request for http://rabobank.nl
redirect (302) to https://rabobank.nl
browser follows redirect
https://rabobank.nl
user connected with HTTPS

MitM attack on such a session


user
MitM
website
user types in rabobank.nl
request for http://rabobank.nl
redirect (302) to https://rabobank.nl
attacker follows redirect
https://rabobank.nl
change HTTPS links to HTTP links
careful user will notice missing s in browser toolbar
server thinks there is nothing wrong!

SSL stripping

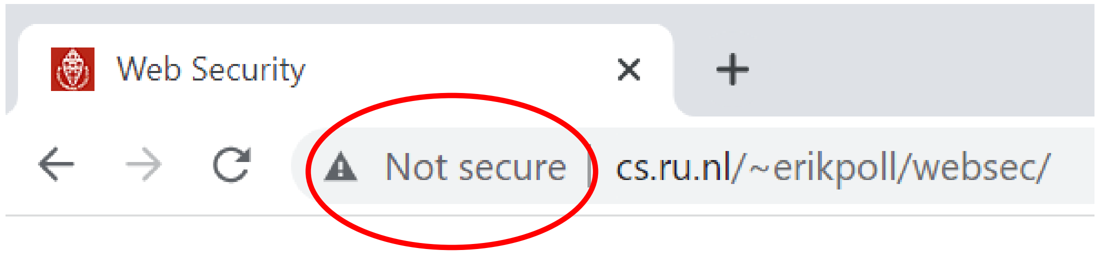
- The MitM attacker
  - strips S from HTTPS in links in traffic from server to user
  - puts this S back in traffic from the user to the server
- The result



- 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

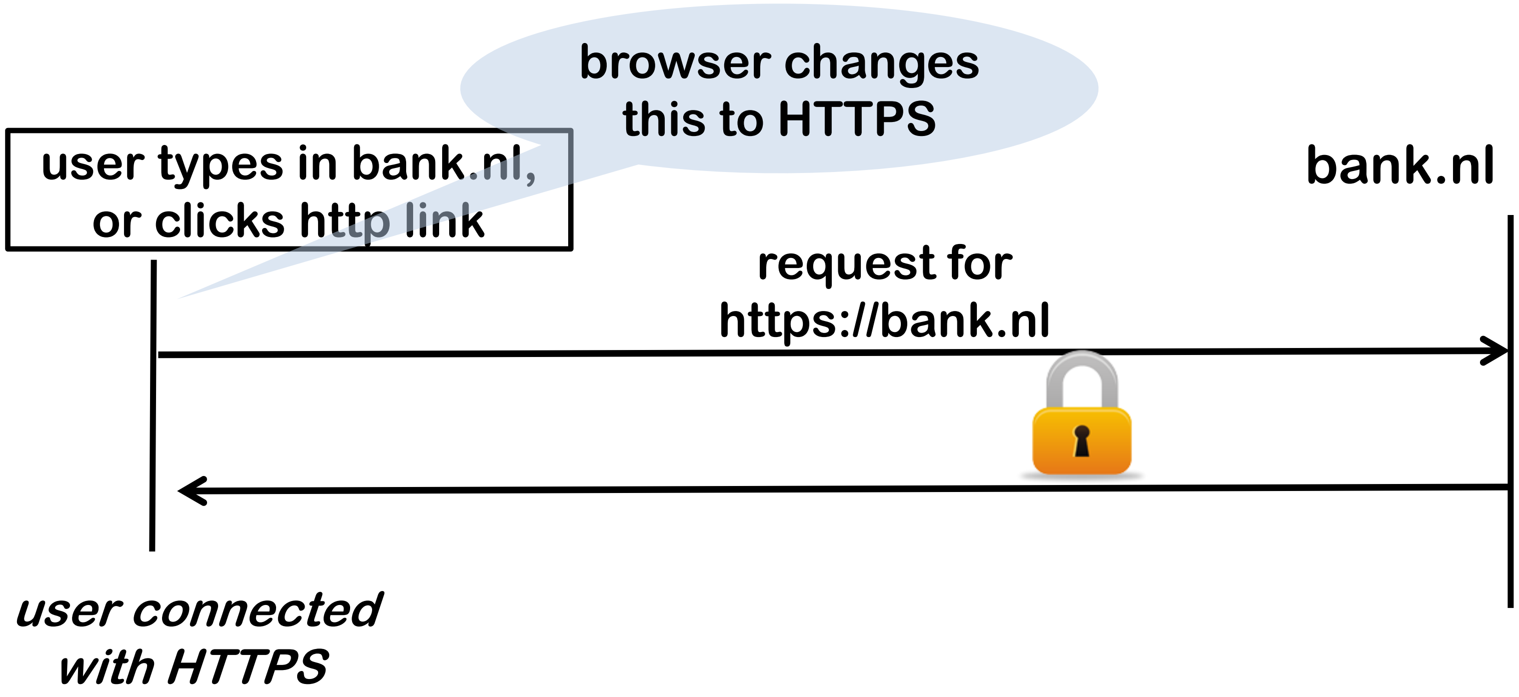
   Eg browser will turn `http://bank.nl/rekening/...`
   into `https://bank.nl/rekening/...`

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


browser changes this to HTTPS
user types in bank.nl, or clicks http link
bank.nl
request for https://bank.nl
user connected with HTTPS

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
  - Tricky in modern browsers given all the 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.
  - Support for HSTS mandatory for Dutch government websites since July 1st, 2023 (Wet digitale overheid)