Software and Web Security 2

Session Management

Recall from last week

- Server and client,
  ie. web application and browser,
  communicate by HTTP requests and responses

- HTTP response can be with GET & POST
  - GET: parameters in URL
  - POST: parameters in HTTP body

Lack of state in HTTP

- HTTP is a stateless protocol
  - it does not remember if there were previous requests
  - bad for web applications (eg. logged in?)

- For most web applications we need the notion of a session:
  a sequence of HTTP requests & responses that belong together

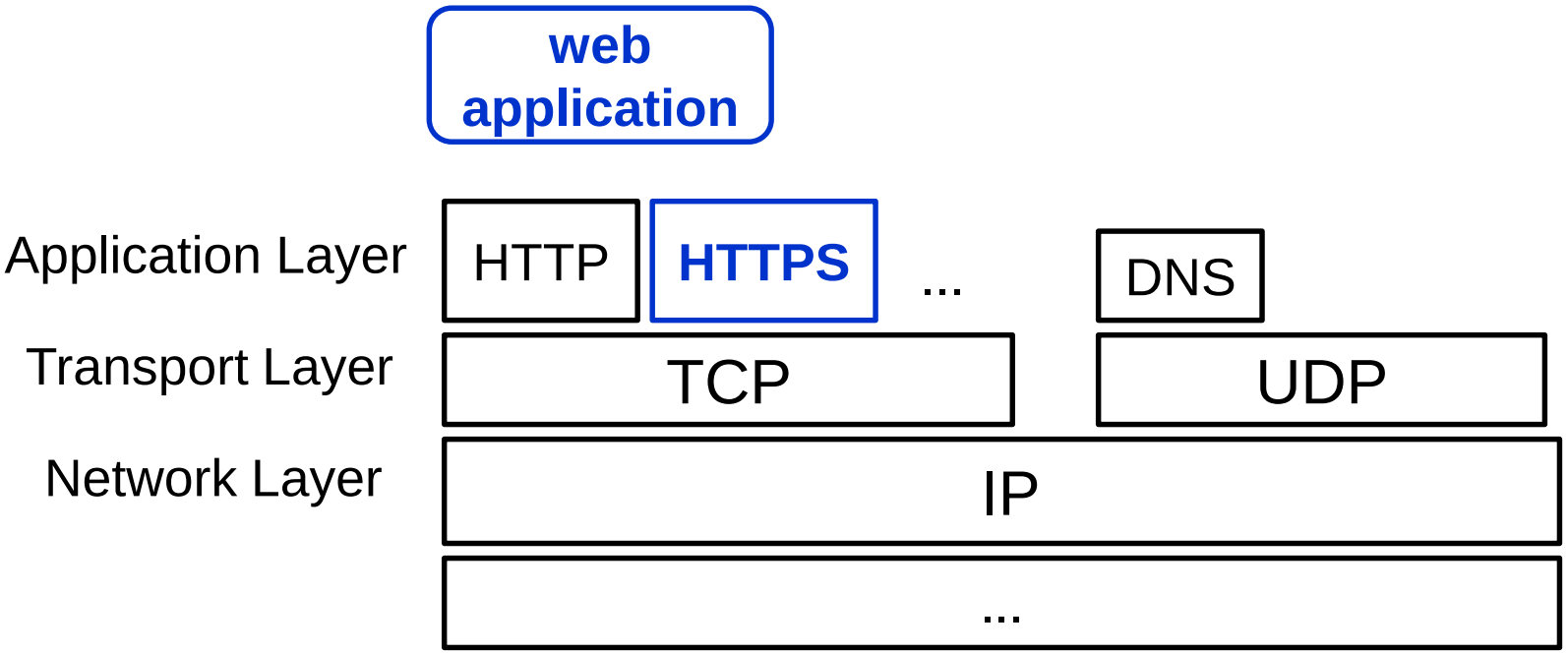
- *Why can't we use IP address for this?*
  - multiple clients can share the same IP address
    eg different users on lilo.science.ru.nl
  - idem for servers
    eg osiris & blackboard hosted at same RU machine
  - clients can change IP address
    or even servers might

session management

Two levels at which sessions can be created

1. by the web application itself
2. at the transport layer, using HTTPS

What is the best solution? Use both!



sessions by the web application
(Section 7.1.4 of book)

sessions managed by the web application

- Web application creates & manages sessions
- Session data is stored at server and associated with a unique session ID
- After session has been created, client is informed of session ID
  - and client attaches session ID to subsequent requests
- Hence server knows about previous requests

- Web application frameworks usually provide built-in support for session management
  - but a web application developers can implement their own

  NB it is better to use existing solutions than inventing your own

  Still, don't underestimate the complexity of using these correctly.

sessions & authentication

The notion of session close tied with authentication

- Eg after logging in with a username & password,
  you will often have certain access rights for the rest of the session

While the session last, *any information that can be used by an attacker to spoof the session (eg a session ID) is just as valuable as the username/password.*

Example: session ID in URL

Web page returned by the server contains links with session ID as extra parameter

```
<html>
Example web page with session IDs in the URL.
The user can now click
<a href="http://demo.net/nextpage.php?sid=1234">here</a>
or
<a href="http://demo.net/anotherpage.php?sid=1234>here</a>
passing on its session id back to the server
wherever we go next.
</html>
```

Example: session ID in hidden parameter

```
<htm>
The form below uses a hidden field
<form method="POST" action= "http://ru.nl/register.php">
   Email: <input type="text" name="Your email address">
   <input type="hidden" name="sid" value="s1234">
   <input type="submit" value="Click here to submit">
</form>
```

Hidden means hidden by default by browser, *not* hidden from a proxy like WebScarab. Browser plugins will show hidden fields and let them be edited.

A hidden form field could also be used to track user preferences, eg

```
<input type="hidden" name="Language" value="Dutch">
```

The same goes for an extra parameter in the URL.

Cookies

- Piece of information that is set by the server & stored by the browser
  - when HTPP response includes **`Set-Cookie`** field in header
  - belongs to some domain, eg www.test.com
  - includes expiry date, domain name, optional path, optional flags
    - eg **`secure`** and **`HTTPOnly`** flags

- Cookie is sent automatically sent back by the browser, for any request to that domain
  - in the **`Cookie`** field of HTTP request
  - effectively, provides state information about the current session

- Cookie can include any type of information
  - sensitive information, such as password, credit card nrs, ..
  - less sensitive information, such as language preferences

  Almost all websites use cookies.

Different ways to provide session ID

Three ways to exchange session IDs

1. encoding it in the URL
2. Hidden form field
3. Cookies

Which is most secure against an attacker eavesdropping on the network traffic?

None is!

Different ways to provide session ID

Pros & cons

1. encoding it in the URL

   Downsides:

   – stored in logs (eg browser history)
   – can be cached & bookmarked
   – visible in the browser location bar

2. Hidden form field

   Marginally better: won't appear in URLs, and cannot be bookmarked, and less likely to be logged

3. Cookies

   Best choice: automatically handled by browser; easier & more flexible.

   But will not work if client does not accept cookies.

Different types of cookies

- non-persistent cookies
  - only stored while current browser session lasts
  - good for sessions
- persistent cookies
  - preserved between browser sessions
  - useful for maintaining user preferences across sessions
  - lousy for privacy...
- **`secure`** cookies
  - only sent over encrypted HTTPS connections

  Encrypting cookies sent over insecure, ie HTTP, connection is pointless.

  Why?

  Attackers can simply replay a stolen encrypted cookie
- **`HTTPonly`** cookies

  Inaccessible to scripts; will be discussed when we do client side scripting

domains, subdomain, and top level domains

The domain in a cookie can be a subdomain of a website,

eg the subdomain **`cs.ru.nl`** of **`ru.nl`**

Complex set of rules govern access across (sub)domains

- subdomains can access cookie for domain, but not vice versa
- subdomains can set cookie for direct superdomain, but not vv

Rationale: subdomains need not trust their superdomain

For top level domains, eg **`.nl`**, there are additional rules,

to prevent **`ru.nl`** from setting a cookie for **`.nl`**

But does all this work as intended for countries that use 3 level domain names?

Eg **`somecompany.co.uk`**, where **`co.uk`** is not a top level domain

Some session attacks

Aim of attacker: steal the session ID

- this can be session cookie, or other form of session ID
- if the victim is logged in, this is just as good as stealing his username and password

- Interception
  - intercept request or response and extract the cookie (eg on unprotected wifi)
- Prediction
  - try to guess a session ID
- Brute Force
  - try many guesses for the session ID, until you get lucky,,,
- Fixation
  - make the victim use a certain session ID

Session ID prediction example

Suppose you can check your grades in blackboard on page

`blackboard.ru.nl/grades.php?s=s776823`

where s776823 is your student number.

Then you could try other student IDs or – better still – the university employee number of the teacher.

Session fixation example

If sessionIDs are passed in URL or hidden form field, then an attacker

1. start a session and obtain a session ID;
2. craft a link (or a webpage) for the victim, and try to get the victim to click that link (or visit that page and click links on it), with that session ID included;
3. the victim now goes to the website using a known session ID;
4. if the victim logs on, and the session ID is not changed, the attacker can abuse the user's rights!

Alternatively, the attacker could craft a webpage, to do this

- eg a webpage that looks just like the login screen of the genuine website.

Session attacks on cookies require client side scripting; we discuss these later.

Making session attacks harder

- use an encrypted HTTPS connection

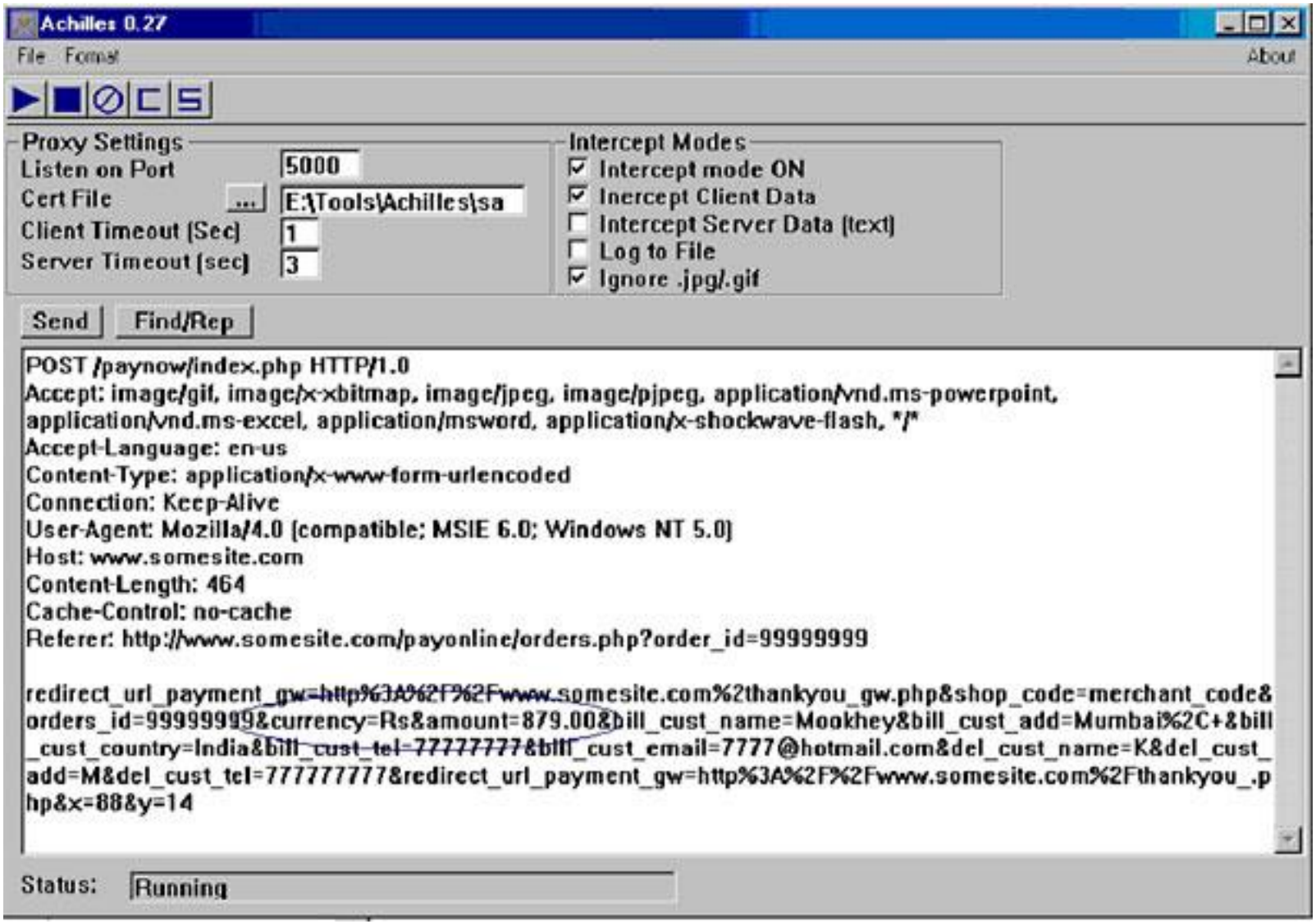
  for *all* requests & responses that include the session ID, not just the login
- Use long enough, random session IDs
- associate session to an IP address

  and disallow the same session ID to be used from a different IP address
  - downside: if legitimate user changes IP address during session his session breaks
- change session ID after any change in privilege level

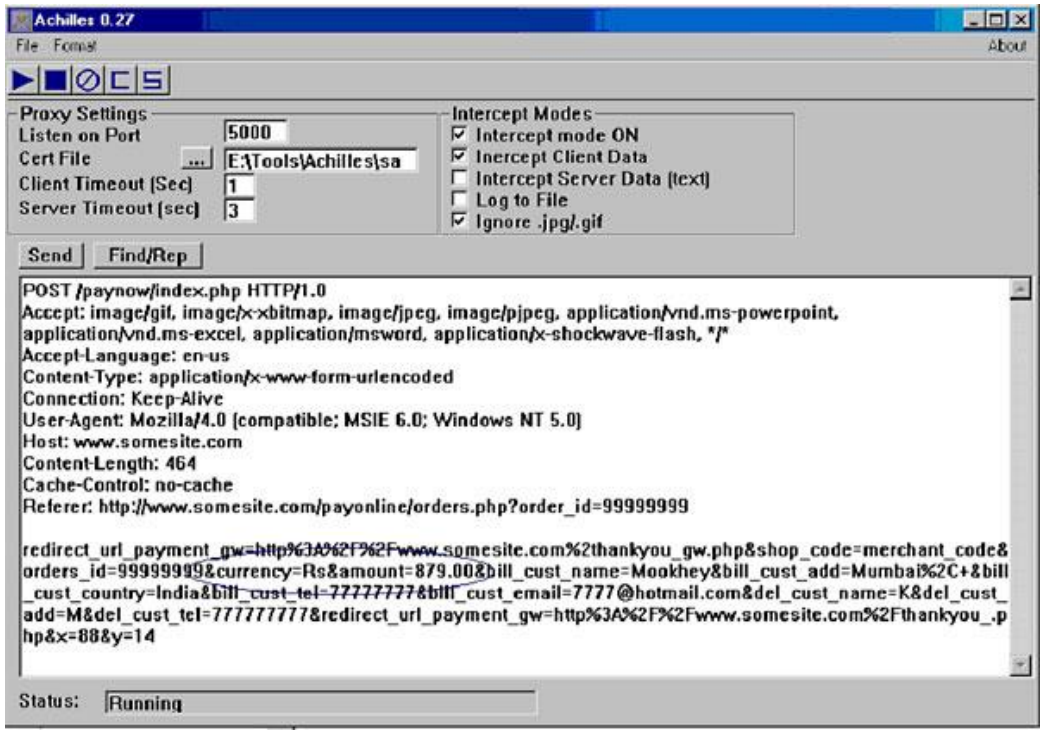
  eg after logging in
- expire sessions

  eg by setting expiration on cookies
- require the clients to re-authenticate, esp. before important actions

What about other information besides a session ID?



Things that can go wrong



Classic security flaw in shopping web site store application: price is recorded in a hidden form field, as shown by the web proxy output above. The client can change this...

MAC

To prevent tampering or spoofing of sessions IDs, cookies, or other data, sent back by the browser, the server can use MAC (Message Authentication Code)

- MAC is like a digital signature, but using (faster) symmetric crypto rather than (slower) public key crypto.

To MAC data, the server

- computes hash of the session ID (or cookie, hidden parameters, or any other info passed to the client);
- encrypts this hash with a symmetric key only server knows, $\{hash(M)\}_{Key}$
- appends MAC to the session ID (or in cookie, or as hidden param).

When info with MAC is returned by client, the server can now check its integrity.

Any data sent to the client that the client should return unaltered to the server, should be MAC-ed : the server should not trust the client!

Alternatively, the server could just send session ID and keep all other important info locally

sessions using HTTPS
(Section 7.1.2 in book)

HTTPS (and SSL,TLS, SSL/TLS, TLS/SSL,..)

HTTPS runs HTTP over TLS/SSL.

- TLS (Transport Layer Security) is new version of SSL (Secure Sockets Layer): TLS version 1.0 is SSL version 3.1
- in practical usage, SSL and TLS are often synonyms,
  eg certificates used for TLS are typically called SSL certificates

TLS/SSL highly configurable, and security guarantees depend on configuration:

- typically, integrity and confidentiality of the session
  - attacker on the network can still see *that* two IP addresses communicate, but not *what*
- often also server authentication
  - ie client authenticates the server
- rarely also client authentication
  - ie server authenticates the client

HTTPS

1. Server sends certificate to client, which
   - includes server's public key **`PK`**
   - is digitally signed by Certificate Authority (CA), or self-signed
   - browsers come pre-configures with a list of trusted CAs
2. Client checks that certificate has not been revoked
   - by requesting Certificate Revocation List (CRL) from CA
3. client authenticates the server
   - by sending random challenge **`nonce`** encrypted with **`PK`**, and checking some response that includes **`nonce`**
4. client and server agree a symmetric session key
   - based on **`nonce`** and some random number chosen by server
5. Subsequent HTTP traffic encrypted *and MACed* with session key
   - encryption for confidentiality, MACing for integrity
   - periodically the session key is refreshed

securing session vs individual request/responses

Subtle but important difference between

- integrity of individual HTTP requests and responses
- integrity of the *session* of HTTP requests and responses

Integrity of the session means that *the sequence (incl. the order) of messages is intact.*

Having only integrity of individual messages would allow replay attacks and other Man-in-the-Middle attacks

where attacker removes, repeats, or reorders some HTTP traffic.

regular vs Extended Validation certificates

- There are no specific requirements on how a CA should check its clients before it issues its certificates
- For Extended Validation (EV) certificates there are.
  - and some auditor checks that CA sticks to these

- Most browsers show padlock for regular certificates, and a green URL in the location bar for EV certificates

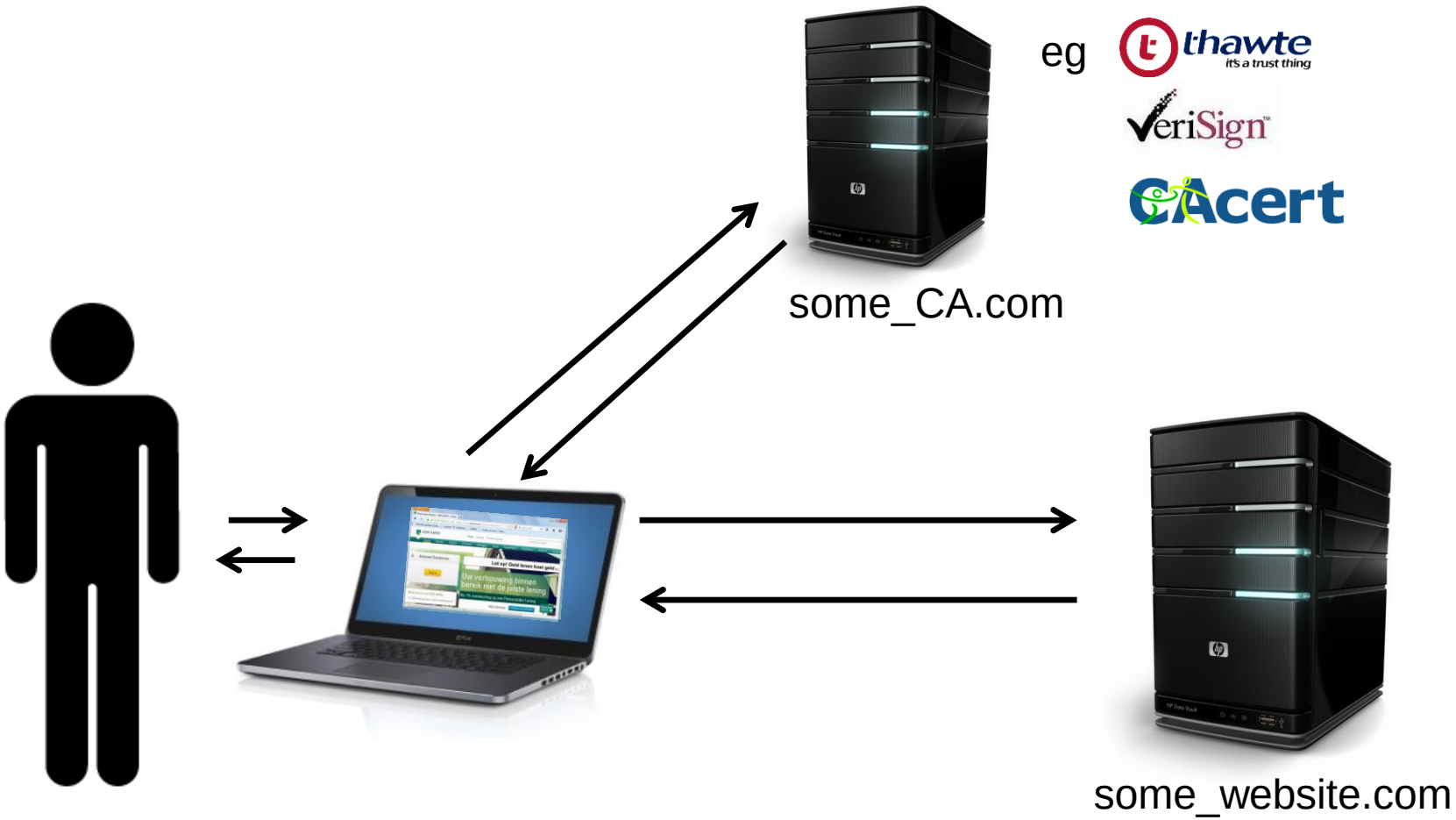
NB the fact that some certificates – or some CAs - are "more secure" than others, raises interesting questions:

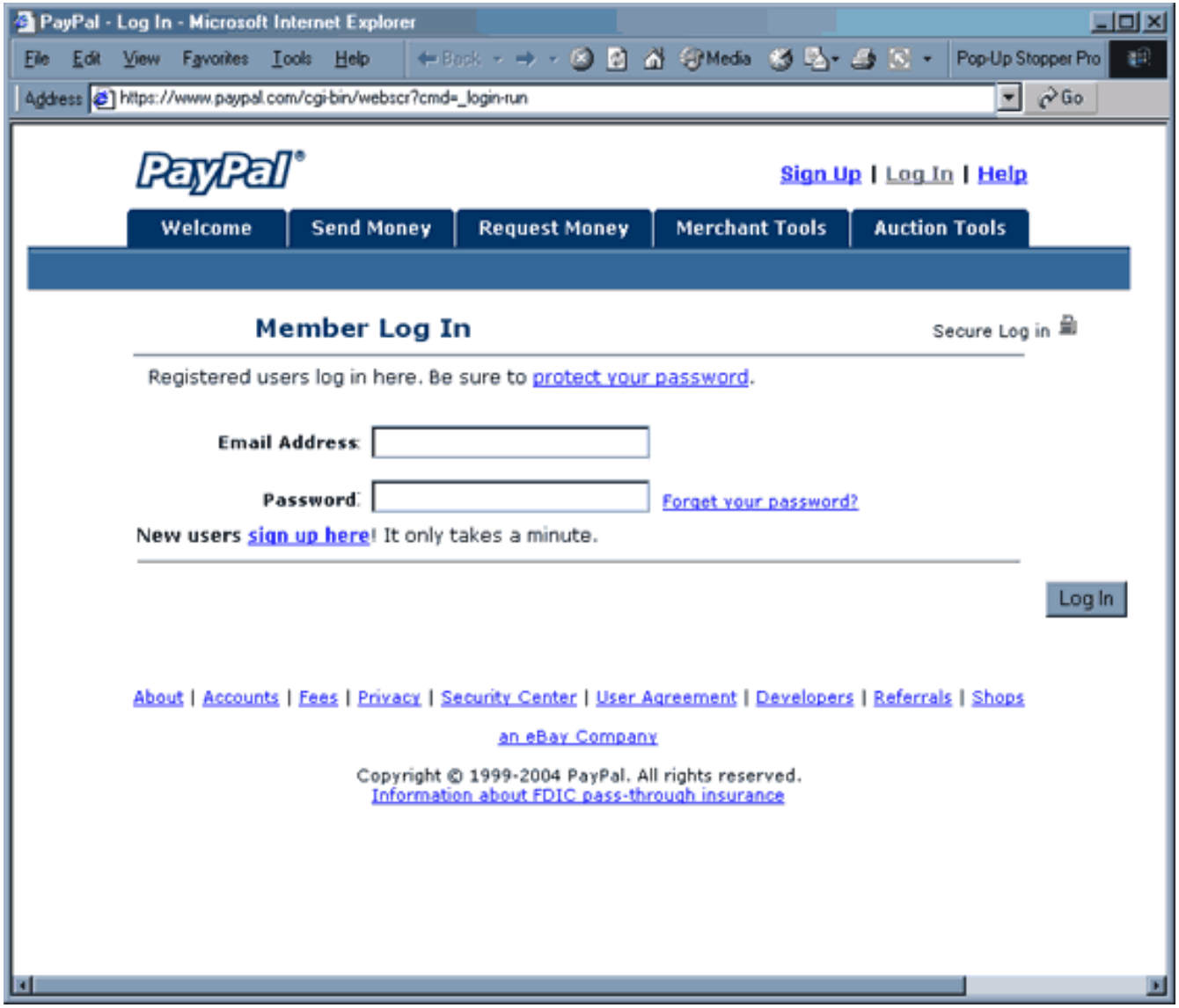
how secure is all this anyway?

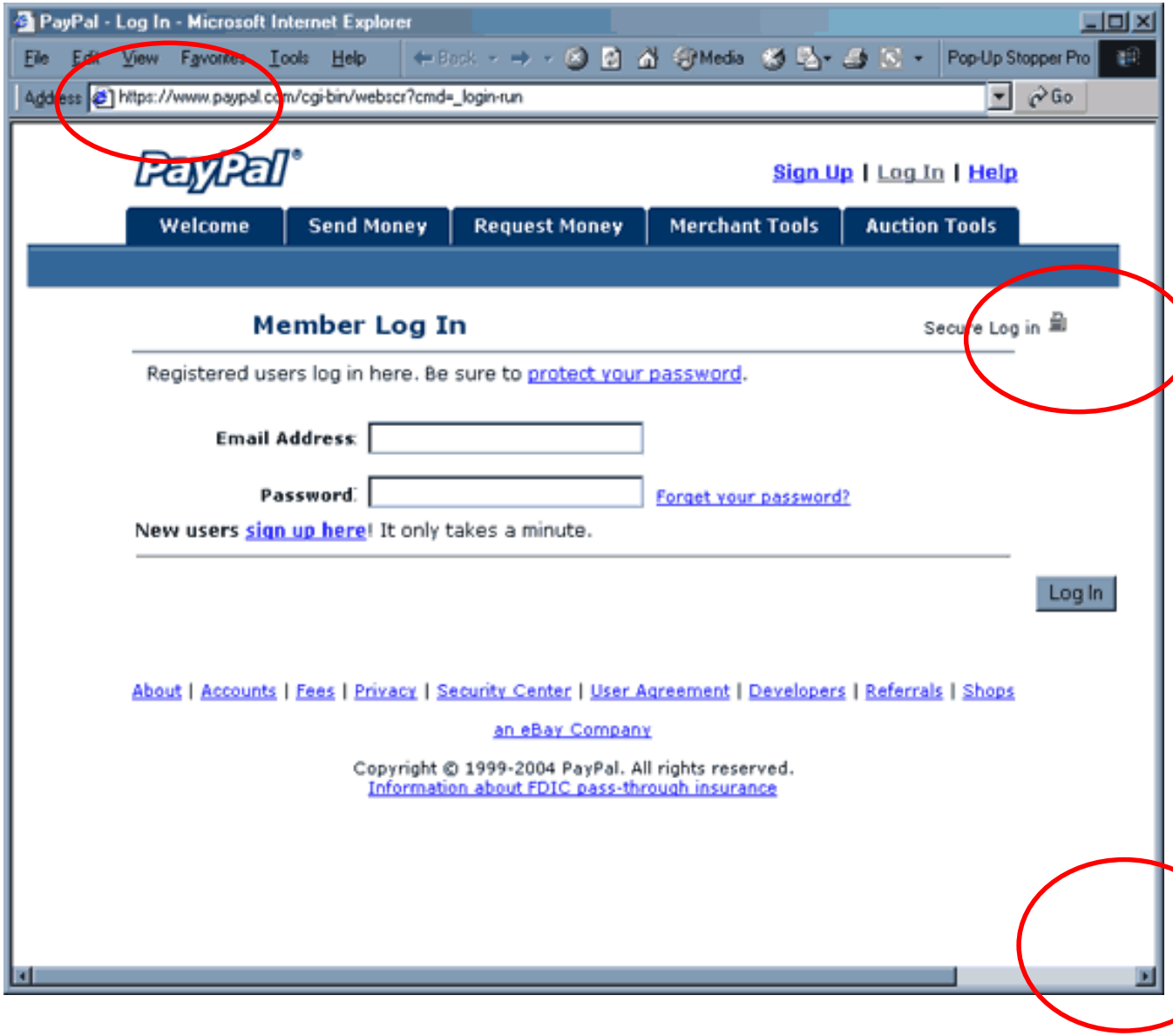
who and what are we trusting?

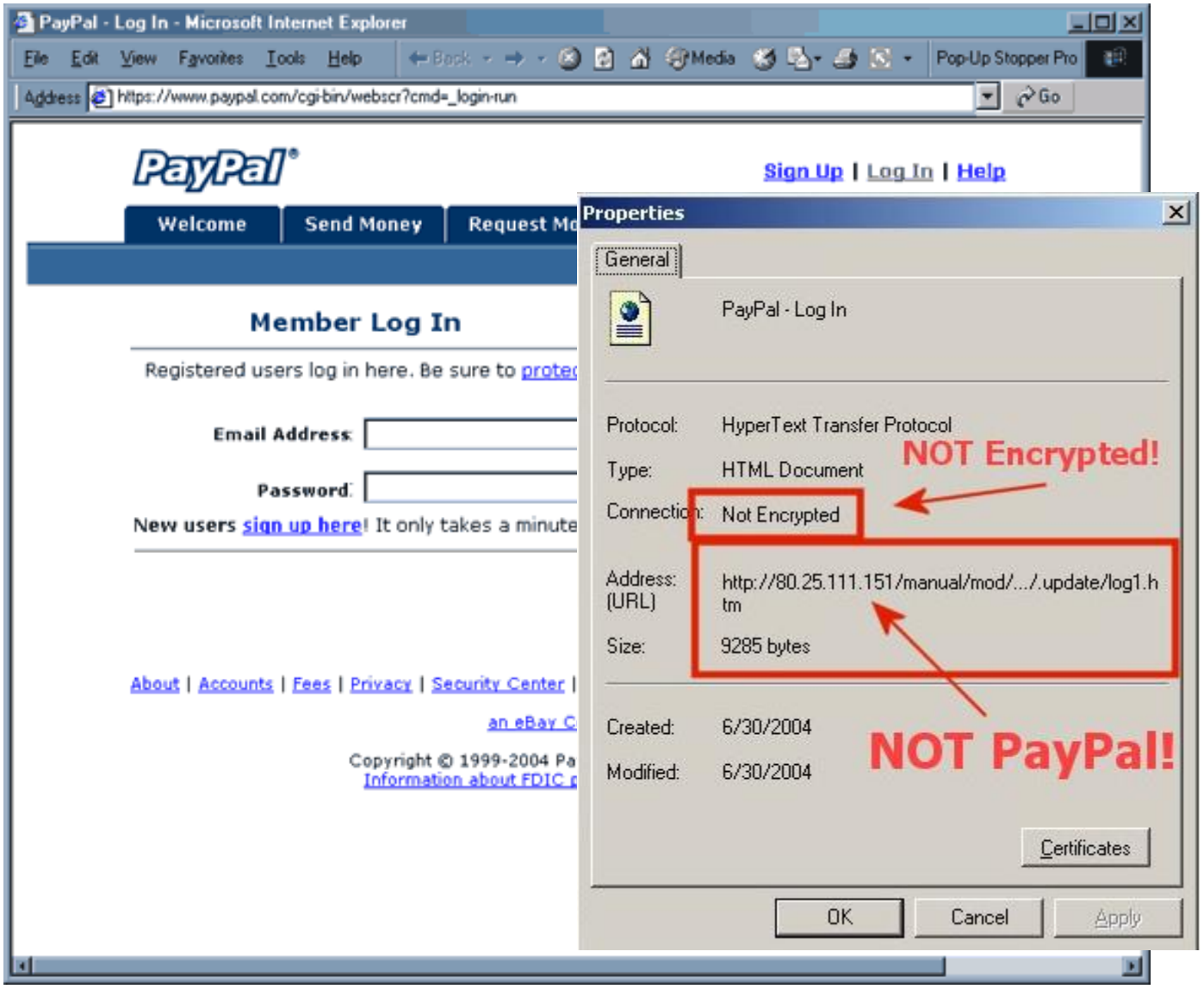
What are we trusting?

What are we trusting?




PayPal - Log In - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Media Pop-Up Stopper Pro
Address https://www.paypal.com/cgi-bin/webscr?cmd=_login-run
Go
PayPal
Sign Up | Log In | Help
Welcome
Send Money
Request Money
Merchant Tools
Auction Tools
Member Log In
Secure Log in
Registered users log in here. Be sure to protect your password.
Email Address:
Password:
Forget your password?
New users sign up here! It only takes a minute.
Log In
About | Accounts | Fees | Privacy | Security Center | User Agreement | Developers | Referrals | Shops
an eBay Company
Copyright © 1999-2004 PayPal. All rights reserved.
Information about FDIC pass-through insurance


PayPal - Log In - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Media Pop-Up Stopper Pro
Address https://www.paypal.com/cgi-bin/webscr?cmd=_login-run
Go
PayPal
Sign Up | Log In | Help
Welcome
Send Money
Request Money
Merchant Tools
Auction Tools
Member Log In
Secure Log in
Registered users log in here. Be sure to protect your password.
Email Address:
Password:
Forget your password?
New users sign up here! It only takes a minute.
Log In
About | Accounts | Fees | Privacy | Security Center | User Agreement | Developers | Referrals | Shops
an eBay Company
Copyright © 1999-2004 PayPal. All rights reserved.
Information about FDIC pass-through insurance


PayPal - Log In - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address https://www.paypal.com/cgi-bin/webscr?cmd=_login-run
PayPal
Sign Up | Log In | Help
Welcome
Send Money
Member Log In
Registered users log in here. Be sure to
Email Address:
Password:
New users sign up here! It only takes a minute
About | Accounts | Fees | Privacy | Security Center |
Copyright © 1999-2004 Pa
Information about FDIC
Properties
General
PayPal - Log In
Protocol: HyperText Transfer Protocol
Type: HTML Document
Connection: Not Encrypted
NOT Encrypted!
Address (URL): http://80.25.111.151/manual/mod/.../.update/log1.htm
Size: 9285 bytes
Created: 6/30/2004
Modified: 6/30/2004
NOT PayPal!
Certificates
OK
Cancel
Apply

URL obfuscation – or malicious URLs

1. Old bug in Internet Explorer: part of the URL following an Esc or null character was not shown in location bar.

   How can attacker abuse this?

   http://paypal.com%01%00@mafia.com

2. In any browser, an attacker can try to confuse the user, eg.

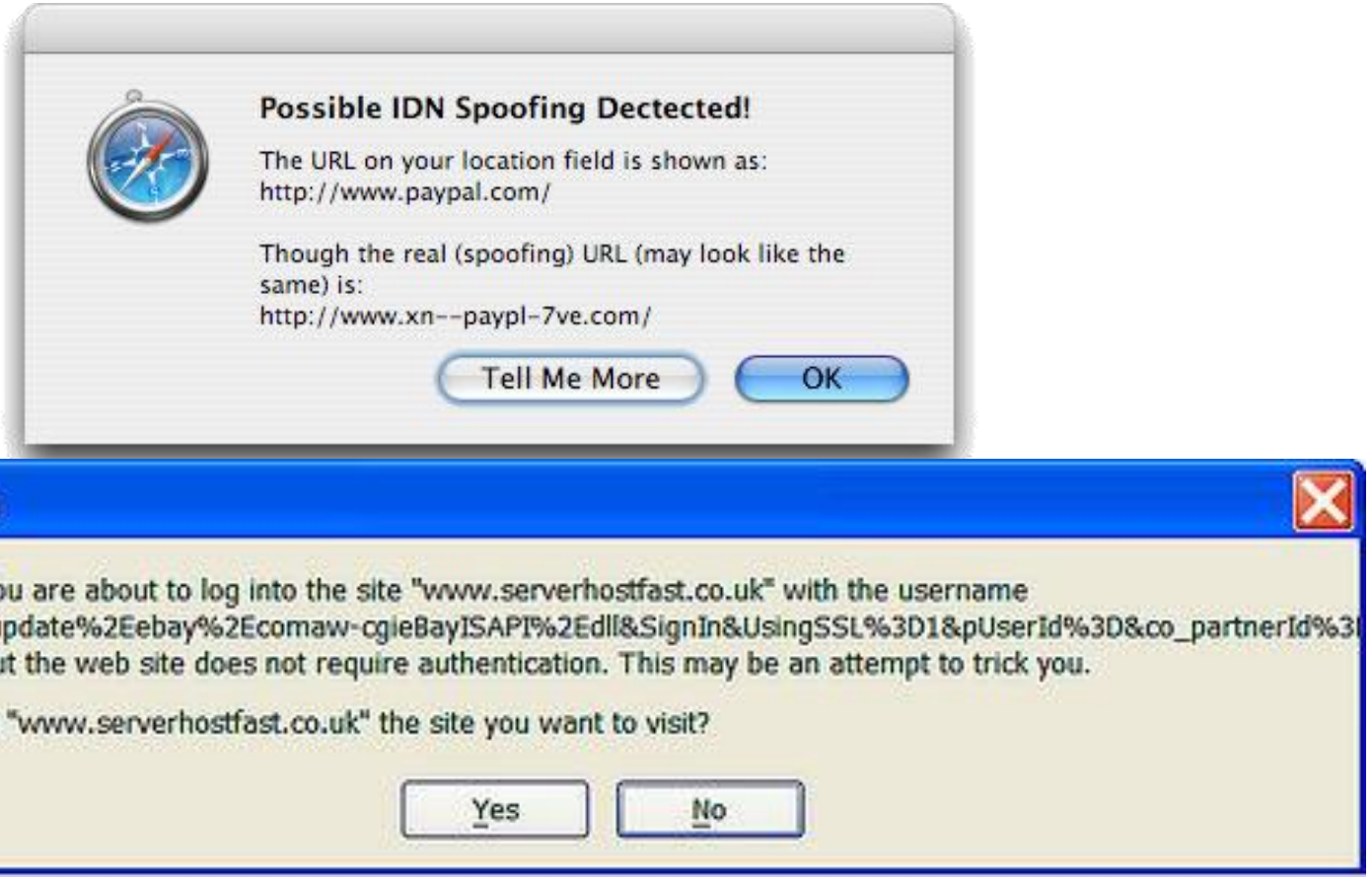
   http://www.visa.com@%32%32%30%2E%36%38%2E%32%31%34%2E%32%31%33

   which translates into **220.68.214.213**

3. Domain names containing Unicode characters
   - eg Cyrillic **p** instead of Latin **p** in **paypal.com**

   Modern browsers use Punycode, an ASCII encoded version of Unicode, which shows this as **www.xn-pypal-4ve.com**

Browser can try to warn user



The bottom message warns that the URL contains a redundant login:password at the start of the URL as in example 2 on previous slide

```
 1  static OSStatus
 2  SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa, SSLBuffer
 3                                   uint8_t *signature, UInt16 signatureLen
 4  {
 5      OSStatus        err;
 6      ...
 7
 8      if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
 9          goto fail;
10      if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
11          goto fail;
12          goto fail;
13      if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
14          goto fail;
15      ...
16
17  fail:
18      SSLFreeBuffer(&signedHashes);
19      SSLFreeBuffer(&hashCtx);
20      return err;
21  }
```

Things that go wrong

About the security content of iOS 7.0.6

This document describes the security content of iOS 7.0.6.

iOS 7.0.6

- **Data Security**

  Available for: iPhone 4 and later, iPod touch (5th generation), iPad 2 and later

  Impact: An attacker with a privileged network position may capture or modify data in sessions protected by SSL/TLS

  Description: Secure Transport failed to validate the authenticity of the connection. This issue was addressed by restoring missing validation steps.

  CVE-ID

  CVE-2014-1266

Things that go wrong: implementing SSL/TLS (1)

Feb 2014: “goto” bug in SSL/TLS in Apple iOS and OSX
CVE-2014-1266
- certificates not correctly checked

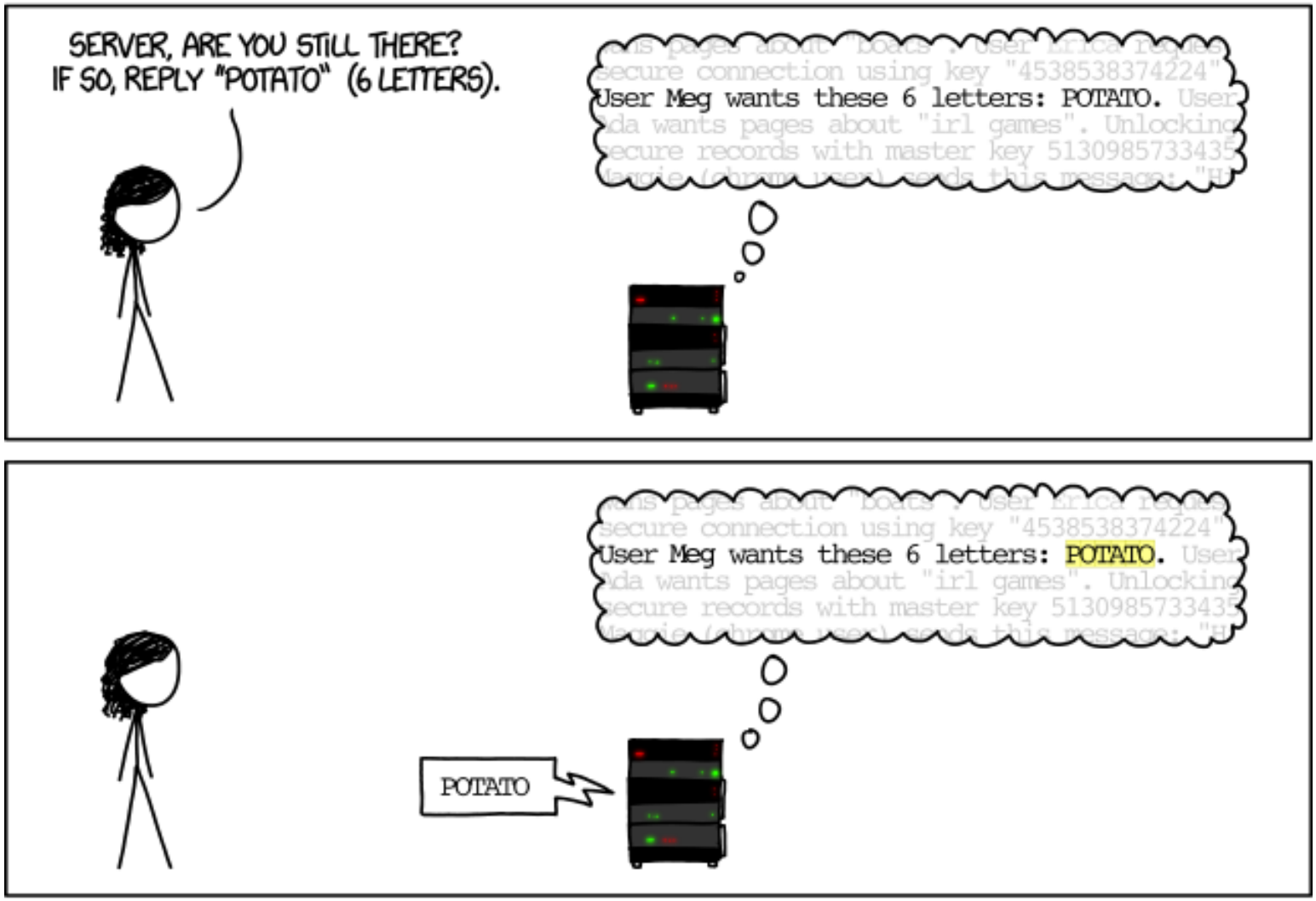
Feb 2014: bug in GnuTLS
CVE-2014-0092
- certificates not correctly checked

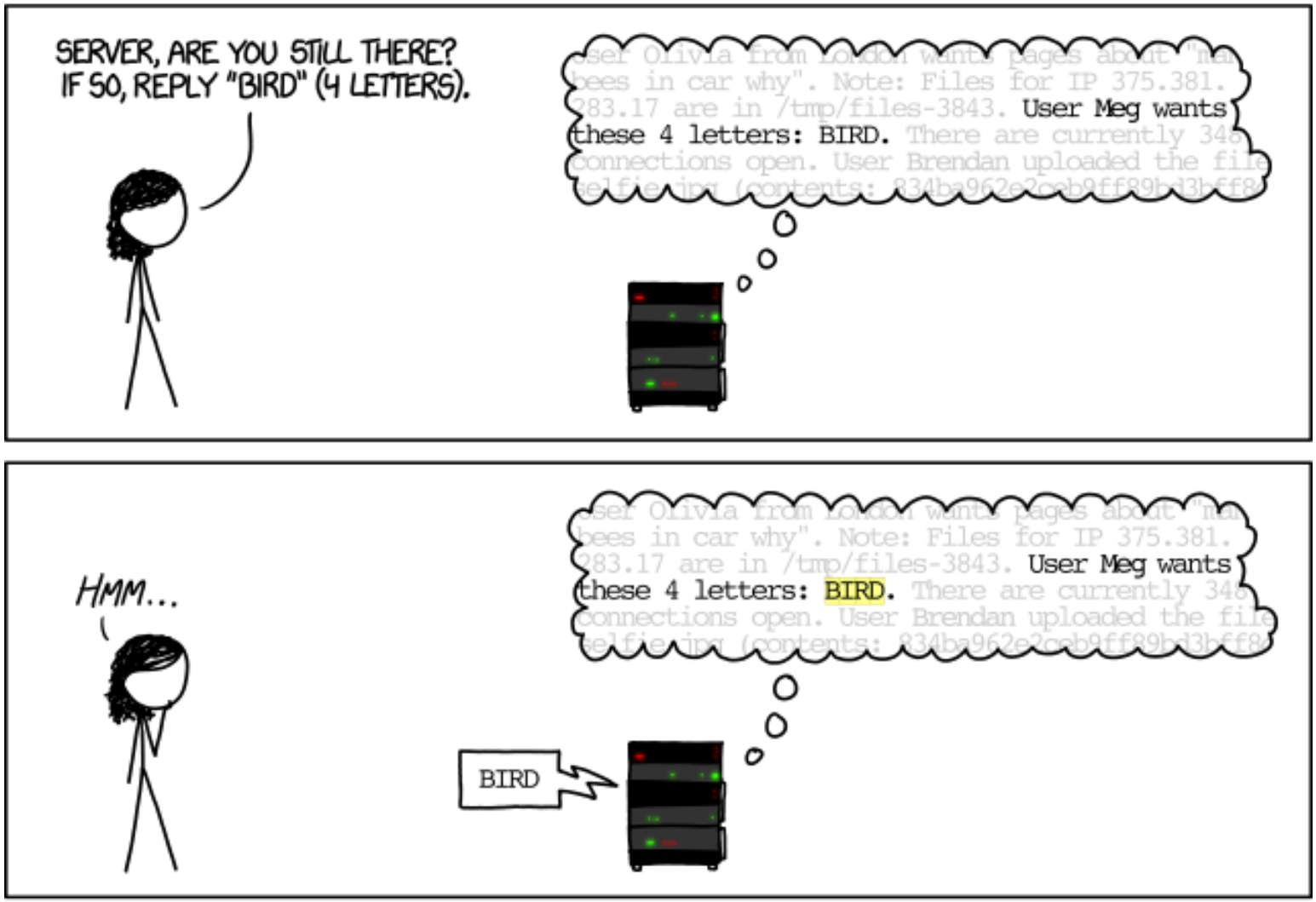
April 2014: heartbleed bug in OpenSSL

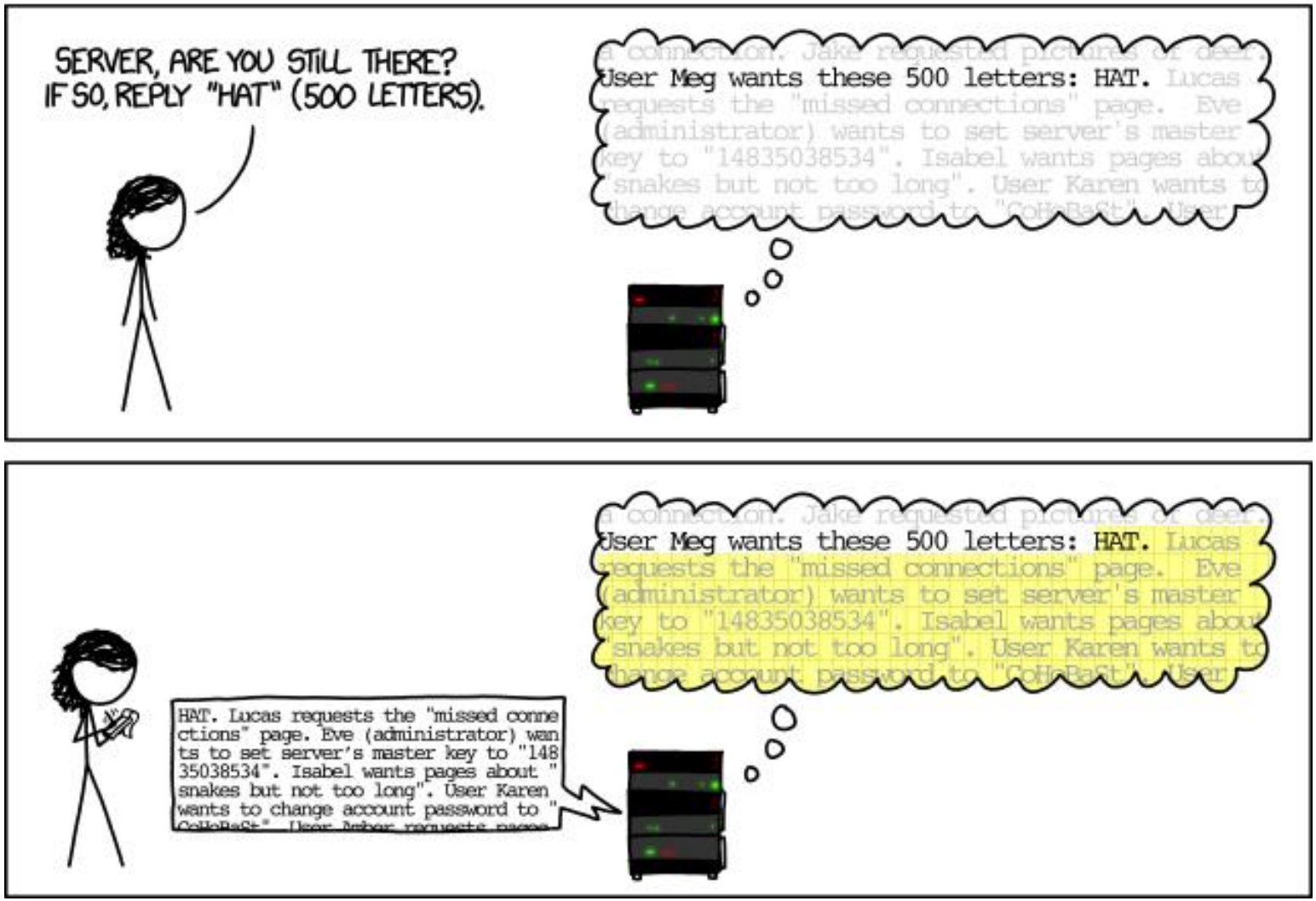
CVE-2014-0160
- buffer overflow allows remote attacker to retrieve data from server

Feb 2015: FREAK attack on Oracle JSSE
CVE-2014-6593
- force fallback to low-strength crypto
- found by Joeri de Ruiter here at the RU, and by researchers at INRIA

HOW THE HEARTBLEED BUG WORKS:




SERVER, ARE YOU STILL THERE? IF SO, REPLY "BIRD" (4 LETTERS).
these 4 letters: BIRD.
HMM...
these 4 letters: BIRD.
BIRD


SERVER, ARE YOU STILL THERE? IF SO, REPLY "HAT" (500 LETTERS).
User Meg wants these 500 letters: HAT.
HAT. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about "snakes but not too long". User Karen wants to change account password to "

more software = more bugs

Ironically, as Heartbleed shows,

any *additional security measure* that you introduce

eg using SSL/TLS

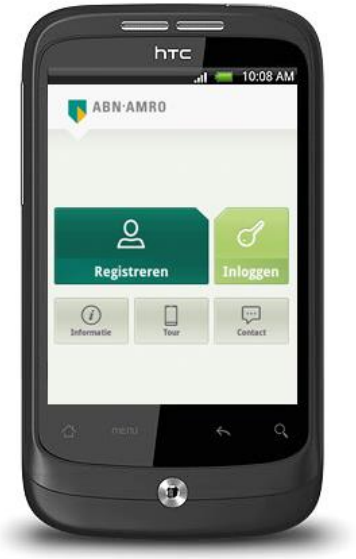
requires *additional software*

that may come with *additional software vulnerabilities*...

Things that go wrong ... (2)

March 2012: ING banking app does not check SSL certificate

December 2012: ABN-AMRO app did not check SSL certificate



Easier to use a generic web browser, which does - or should do - HTTPS checks correctly, rather than implementing SSL/TLS support yourself.

Other things that go wrong ... (3)

Certificate Authority DigiNotar was hacked in 2011.

Fake certificates for google.com were issued.

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

Fortunately, the certificate in the chip in your passport or ID card was not issued by DigiNotar.

What are we trusting?

What are we trusting when we use HTTPS?

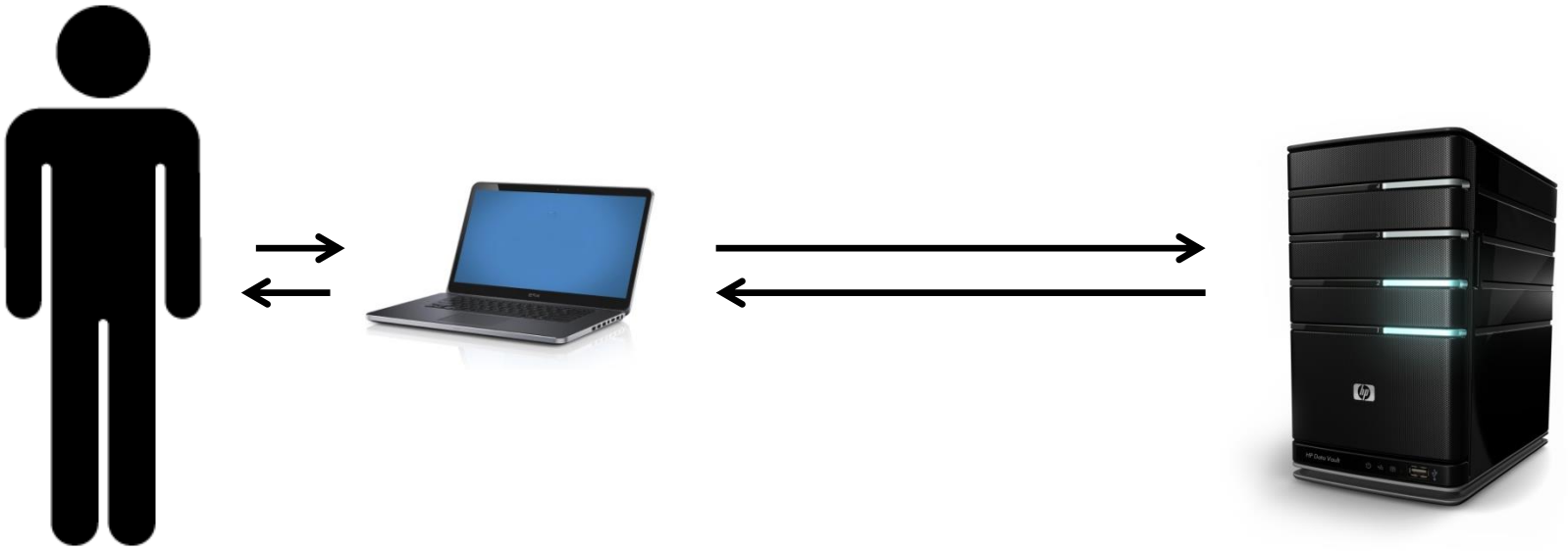
- that web server protects its private key
- that browser checks the certificate correctly, including the CRL
- that user checks for non-ASCII characters in URL, or browser warns about these
- that user check the certificates,
- that CA is not compromised
- that there is no malware in our browser or on our machine that is faking the display
- ....

The Trusted Computing Base (TCB) includes the web server, the browser, the underlying operating systems, the CA, ...

Not to mention the user, the sys admin of the server, the sys admin of the CA, the auditors that audit the CA, ...

Securing the last 30 centimeter...

We can secure connections between computers 1000s of miles apart,
eg using TLS/SSL,
but the remaining 30 cm between user and laptop remain a problem



Using encryption on the Internet is the equivalent of arranging an armored car to deliver credit card information from someone living in a cardboard box to someone living on a park bench.

Gene Spafford

The Trusted Computing Base (TCB) is ***huge***, with all software and hardware at the client, server, and certificate authority.

To do

- Do WebGoat 1a,2a,3a,4a & hackme lvl0 – hand in the lvl0
- Read 7.1.2 and 7.1.4
  - NB we skipped 7.1.3 for now
- check out some certificates in your web browser
  for some sites that use HTTPS
- check out some cookies in your web browser
  eg using a cookie plugin