Securing Internet Communication

CS 161 - Computer Security

Profs. Vern Paxson & David Wagner

TAs: John Bethencourt, Erika Chin, Matthew Finifter, Cynthia Sturton, Joel Weinberger

http://inst.eecs.berkeley.edu/~cs161/

March 15, 2010

Today's Lecture

- Applying crypto technology in practice
- Goal #1: overview of the most prominent Internet security protocols
  - SSL/TLS: transport-level (process-to-process) ala' TCP
  - DNSSEC: securing domain name lookups
  - (Others: SSH, and to a lesser extent, IPSEC)
  - Issues that arising in securing these
- Goal #2: cement understanding of crypto building blocks & how they're used together

Building Secure End-to-End Channels

- *End-to-end* = communication protections achieved all the way from originating client to intended server
  - With no need to trust intermediaries
- Dealing with threats:
  - Eavesdropping?
    - Encryption (including session keys)
  - Manipulation (injection, MITM)?
    - Integrity (use of a MAC); replay protection
  - Impersonation?
    - Signatures

End-to-End ⇒ Powerful Protections

- Attacker runs a sniffer to capture our WiFi session?
  - (maybe by breaking crummy WEP security)
  - Encrypted communication is unreadable
    - No problem!
- DNS cache poisoning?
  - Client goes to wrong server
  - Detects impersonation
    - No problem!
- Attacker hijacks our connection, injects new traffic
  - Data receiver rejects it due to failed integrity check
    - No problem!

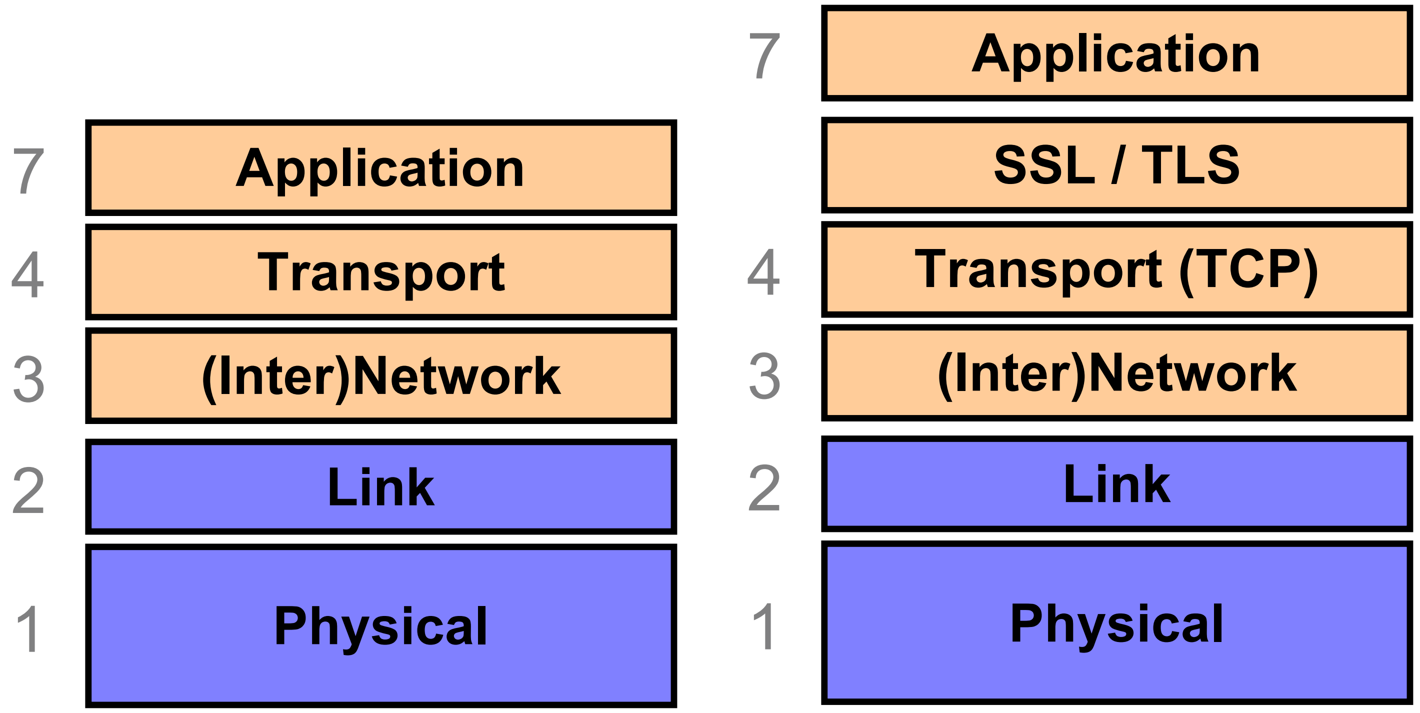
Powerful Protections, con’t

- DHCP spoofing?
  - Client goes to wrong server
  - Detects impersonation
    - No problem!
- Attacker manipulates routing to run us by an eavesdropper or take us to the wrong server?
  - They can’t read; we detect impersonation
    - No problem!
- Attacker slips in as a Man In The Middle?
  - They can’t read, they can’t inject
  - They can’t even replay previous encrypted traffic
  - **No problem!**

Building A Secure End-to-End Channel: SSL/TLS

- SSL = Secure Sockets Layer (predecessor)
- TLS = Transport Layer Security (standard)
  - Both terms used interchangeably
- Notion: provide means to secure *any* application that uses TCP

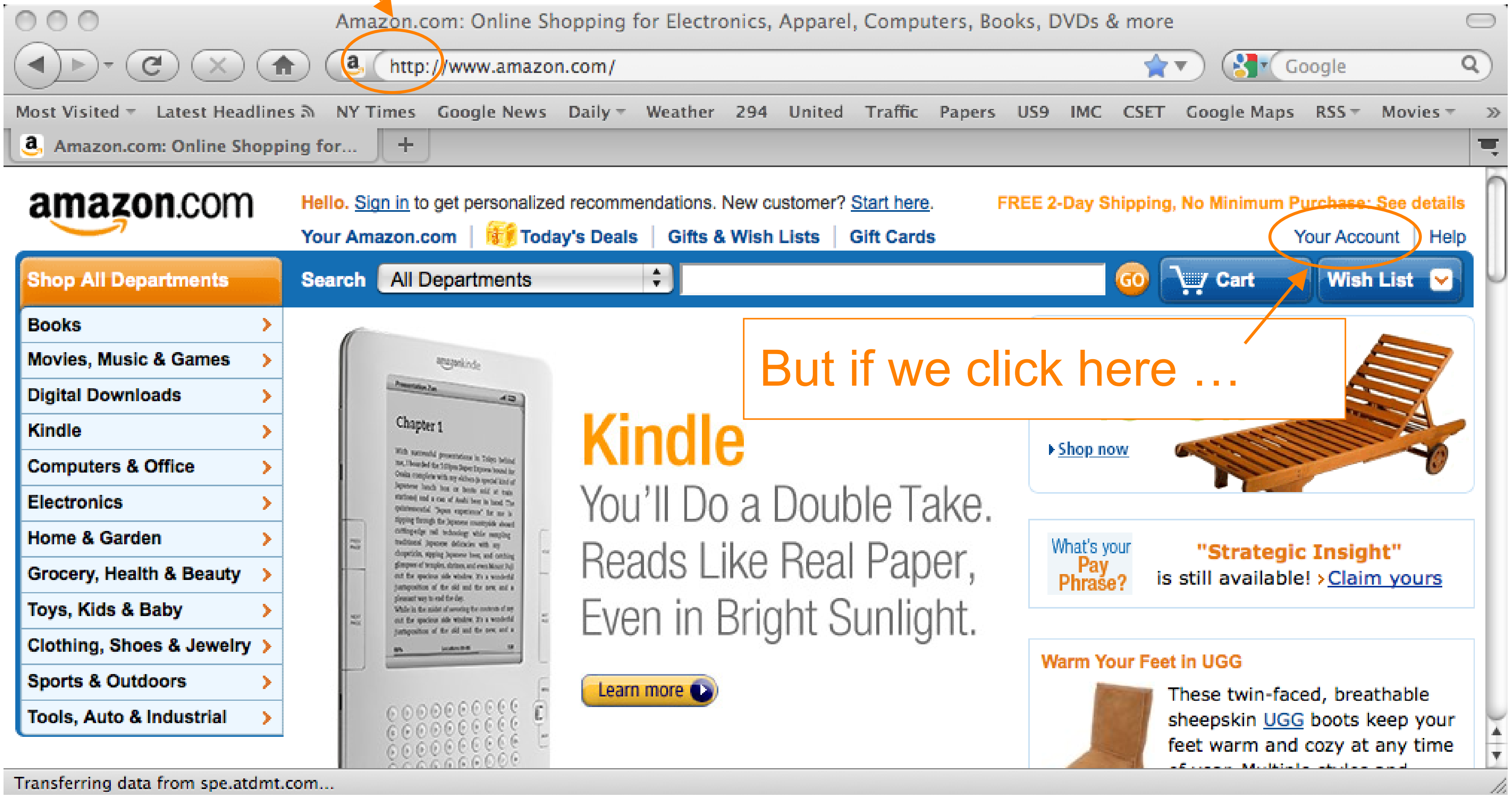
SSL/TLS In Network Layering



Building A Secure End-to-End Channel: SSL/TLS

- SSL = Secure Sockets Layer (predecessor)
- TLS = Transport Layer Security (standard)
  - Both terms used interchangeably
- Notion: provide means to secure *any* application that uses TCP
  - Secure = encryption/confidentiality + integrity + authentication (of server, but *not* of client)
  - E.g., puts the ‘s’ in “https”

Regular web surfing - http: URL



Amazon.com: Online Shopping for Electronics, Apparel, Computers, Books, DVDs & more

http://www.amazon.com/

Most Visited | Latest Headlines | NY Times | Google News | Daily | Weather | 294 | United | Traffic | Papers | US9 | IMC | CSET | Google Maps | RSS | Movies

Amazon.com: Online Shopping for...

Hello. Sign in to get personalized recommendations. New customer? Start here.

FREE 2-Day Shipping, No Minimum Purchase: See details

Your Amazon.com | Today's Deals | Gifts & Wish Lists | Gift Cards | Your Account | Help

Shop All Departments | Search | All Departments | GO | Cart | Wish List

- Books
- Movies, Music & Games
- Digital Downloads
- Kindle
- Computers & Office
- Electronics
- Home & Garden
- Grocery, Health & Beauty
- Toys, Kids & Baby
- Clothing, Shoes & Jewelry
- Sports & Outdoors
- Tools, Auto & Industrial

Kindle

You'll Do a Double Take. Reads Like Real Paper, Even in Bright Sunlight.

Learn more

Shop now

What's your Pay Phrase? "Strategic Insight" is still available! Claim yours

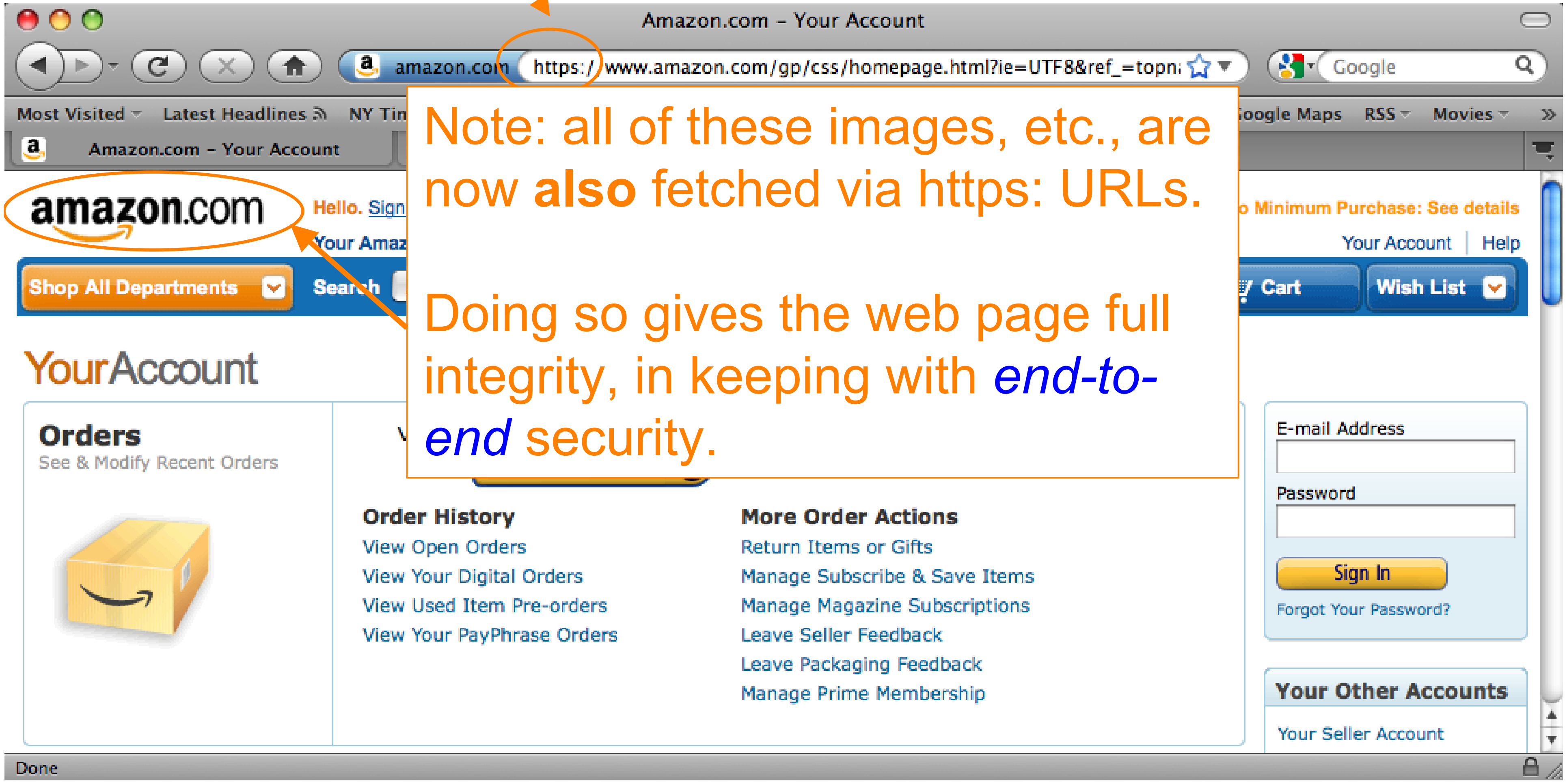
Warm Your Feet in UGG

These twin-faced, breathable sheepskin UGG boots keep your feet warm and cozy at any time

Transferring data from spe.atdmt.com...

But if we click here …

Web surfing with TLS/SSL - https: URL



Note: all of these images, etc., are now **also** fetched via https: URLs.

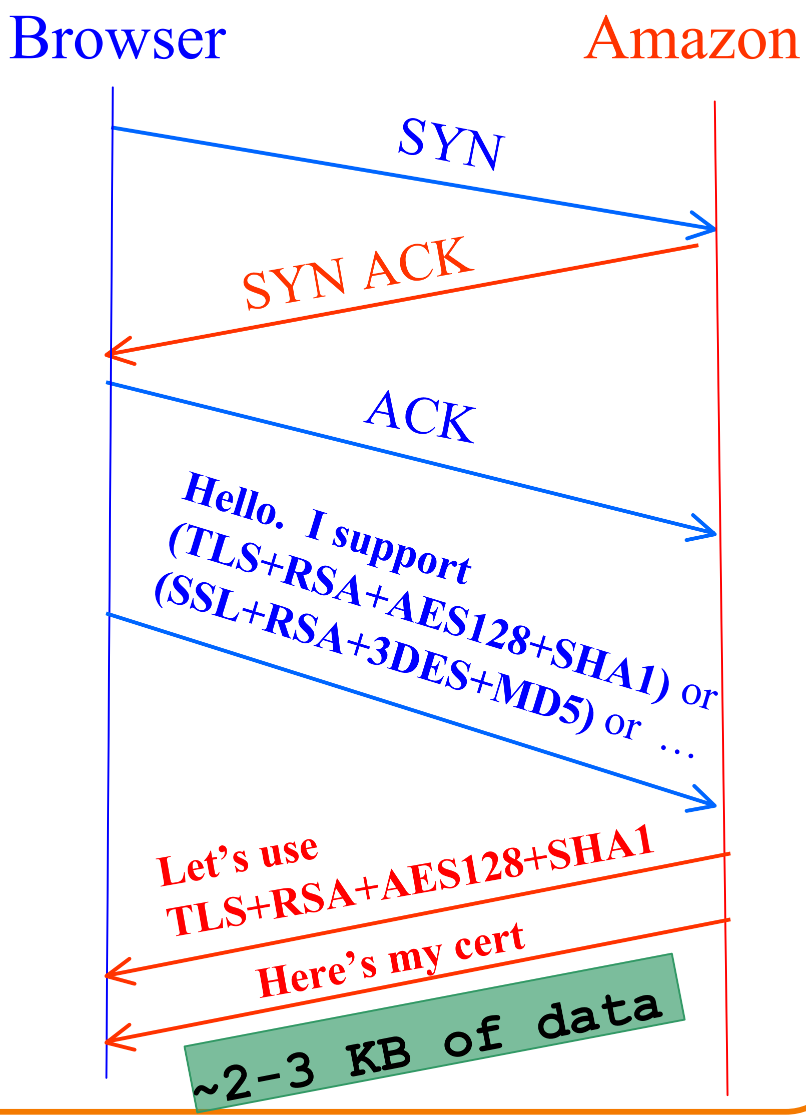
Doing so gives the web page full integrity, in keeping with *end-to-end* security.

Building A Secure End-to-End Channel: SSL / TLS

- SSL = Secure Sockets Layer (predecessor)
- TLS = Transport Layer Security (standard)
  - Both terms used interchangeably
- Notion: provide means to secure *any* application that uses TCP
  - Secure = encryption/confidentiality + integrity + authentication (of server, but not of client)
  - E.g., puts the ‘s’ in “https”
- API similar to “socket” interface used for regular network programming
  - Fairly easy to convert an app to be secured

`HTTPS` Connection (SSL / TLS)

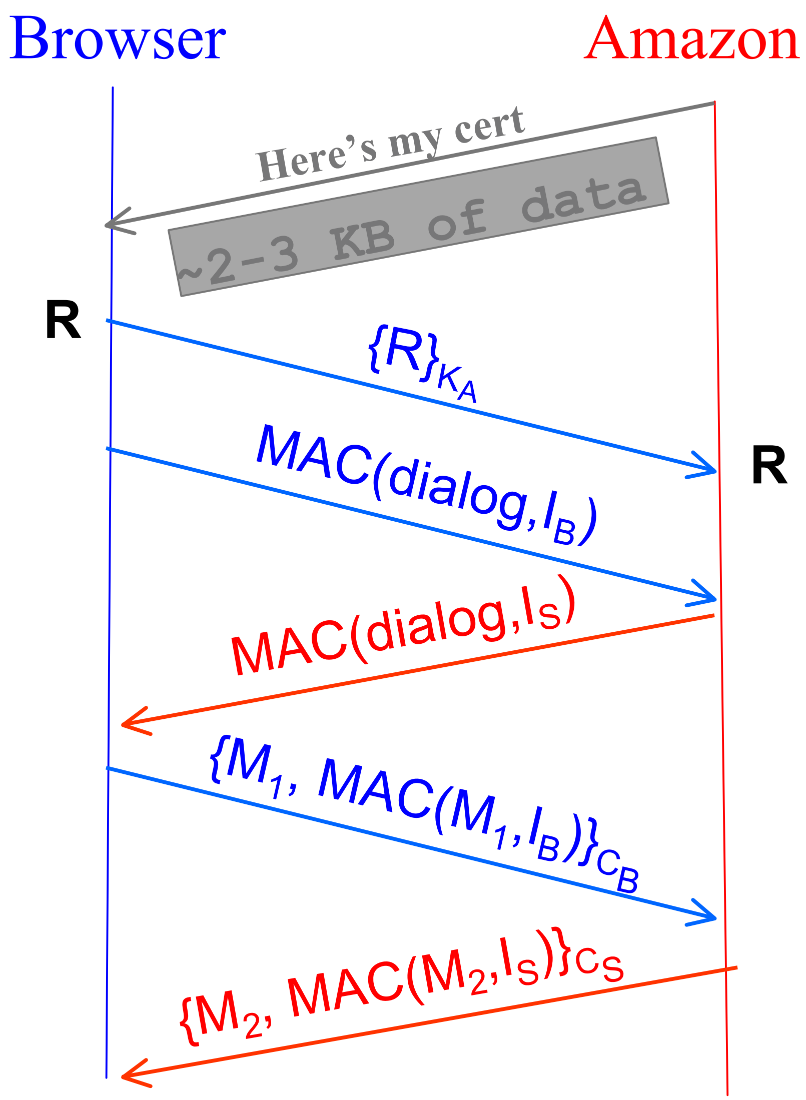
- Browser (client) connects via TCP to Amazon's **`HTTPS`** server
- Client sends over list of crypto protocols it supports
- Server picks protocols to use for this session
- Server sends over its certificate
- (all of this is in the clear)
- ***Client now validates cert***



Browser
Amazon
SYN
SYN ACK
ACK
Hello. I support (TLS+RSA+AES128+SHA1) or (SSL+RSA+3DES+MD5) or …
Let's use TLS+RSA+AES128+SHA1
Here's my cert
~2-3 KB of data

`HTTPS` Connection (SSL / TLS), con’t

- Browser constructs a long (2048 bits) random string R
- Browser sends R encrypted using Amazon’s public key K_A
- From R browser & server derive pairs of symm. *cipher keys* (C_B, C_S) and MAC *integrity keys* (I_B, I_S)
  - One pair to use in each direction
- Browser & server exchange MACs computed over entire dialog so far
- Browser displays
- All subsequent communication encrypted w/ symmetric cipher (e.g., AES128) cipher keys, MACs
  - Messages also numbered to thwart replay attacks



Inside the Server's Certificate

- **Name** associated with cert (e.g., www.amazon.com)
- Amazon's **public key** (e.g., 2048 bits for RSA)
- A bunch of auxiliary info (physical address, type of cert, expiration time)
- Name of certificate's **issuer** (e.g., Verisign)
- Optional URL to *revocation center* to check for revoked certs
- A public-key **signature** of a hash (SHA-1) of all this
  - Constructed using the issuer's private RSA key
  - Call this signature **S**

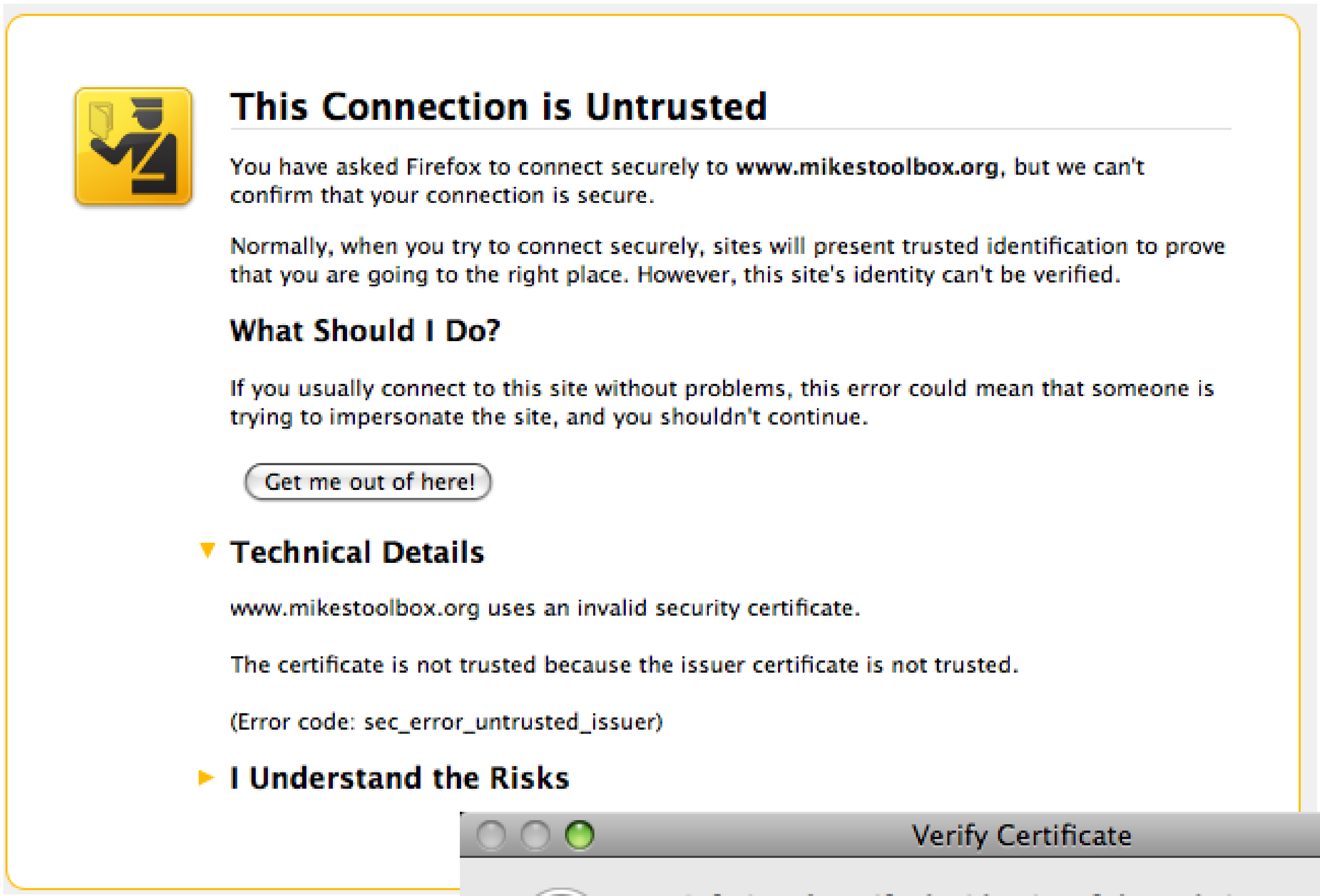
Validating Amazon's Identity

- Browser compares *name* in cert with that in URL
  - Note: this provides an end-to-end property (as opposed to say a cert associated with an IP address)
- Browser accesses *<u>separate</u>* cert belonging to the **issuer**
  - These are hardwired into the browser - **trusted!**
- Browser applies issuer's public key to invert signature **S**, obtaining hash of what issuer signed
  - Compares with its own SHA-1 hash of Amazon's cert
- Assuming hashes match, now have high confidence it's indeed Amazon …
  - ***assuming signatory is trustworthy***

= assuming didn't lose private key; assuming didn't sign thoughtlessly

Validating Amazon’s Identity, con’t

- Browser retrieves cert belonging to the **issuer**
  - These are hardwired into the browser - **trusted!**
- What if browser can’t find a cert for the issuer?


This Connection is Untrusted

You have asked Firefox to connect securely to **www.mikestoolbox.org**, but we can't confirm that your connection is secure.

Normally, when you try to connect securely, sites will present trusted identification to prove that you are going to the right place. However, this site's identity can't be verified.

What Should I Do?

If you usually connect to this site without problems, this error could mean that someone is trying to impersonate the site, and you shouldn't continue.

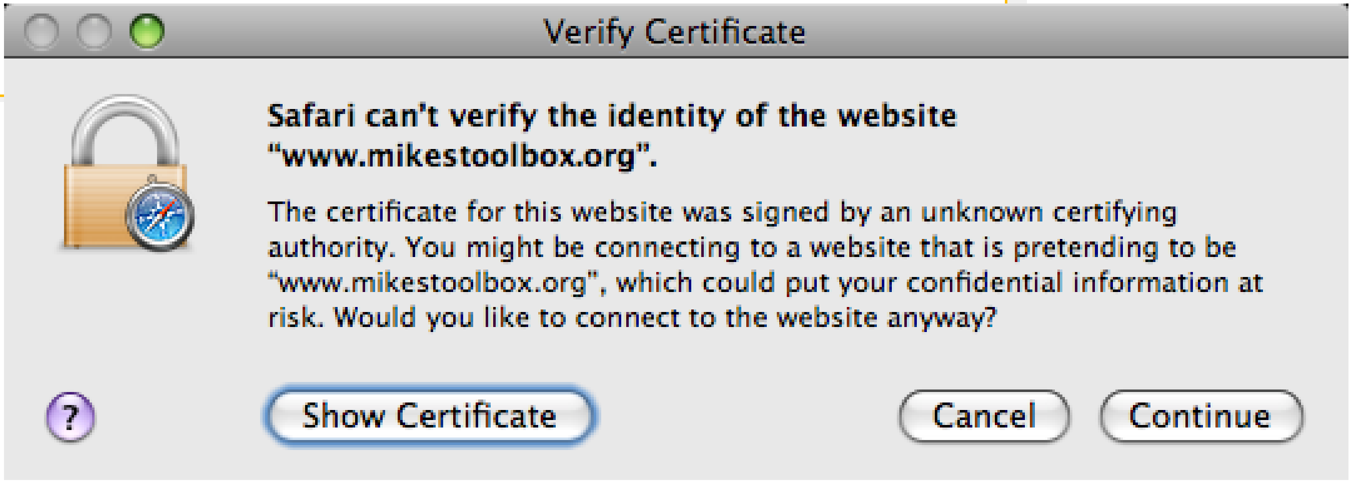
▼ Technical Details

www.mikestoolbox.org uses an invalid security certificate.

The certificate is not trusted because the issuer certificate is not trusted.

(Error code: sec_error_untrusted_issuer)

▶ I Understand the Risks



Safari can't verify the identity of the website "www.mikestoolbox.org".

The certificate for this website was signed by an unknown certifying authority. You might be connecting to a website that is pretending to be "www.mikestoolbox.org", which could put your confidential information at risk. Would you like to connect to the website anyway?

Validating Amazon’s Identity, con’t

- Browser retrieves cert belonging to the **issuer**
  - These are hardwired into the browser - **trusted!**
- What if browser can’t find a cert for the issuer?
- If it can’t find the cert, then warns the user that site has not been verified
  - Note, can still proceed, just without authentication
- Q: Which end-to-end security properties do we lose if we incorrectly trust that the site is whom we think?
- A: **All of them!**
  - Goodbye confidentiality, integrity, authentication
  - Attacker can read everything, modify, impersonate

SSL / TLS Limitations

- Properly used, SSL / TLS provides powerful end-to-end protections
- So why not use it for *everything*??
- Issues:
  - Cost of public-key crypto
    - Can buy hardware to accelerate, but $$
    - Note: *symmetric* key crypto on modern hardware is non-issue
  - Hassle of buying/maintaining certs (fairly minor)
  - DoS amplification
    - Client can force server to undertake public key operations
    - But: requires established TCP connection, and given that, there are other juicy targets like back-end databases
  - Integrating with other sites that don't use HTTPS
  - **Latency**: extra round trips $\Rightarrow$ pages take longer to load

SSL / TLS Limitations, con’t

- Problems that SSL / TLS does **not** take care of ?
- TCP-level denial of service
  - SYN flooding
  - RST injection
    - (but does protect against data injection!)
- SQL injection / XSS / server-side coding/logic flaws
- Browser coding/logic flaws
- User flaws
  - Weak passwords
  - Phishing
- Any service not running over TCP
  - Such as …. ?