

# TLS: Transport Layer Security



# The Early History of the Web

- It's 1994
- Netscape has the first commercial web browser (which they're giving away) and the first commercial web server, which they want to sell
- They want e-commerce to exist, but for that to happen people have to feel secure sending their credit card numbers over the Internet
- 👉 People were already worried about hackers
- Answer: encryption
- But how? They brought in a respected cryptographer, Taher Elgamal, to answer that.

# Three Obvious—but Impossible—Approaches

- Packet-layer encryption, similar to what became IPsec
- Transport-layer encryption, similar to the older SP4 protocol
- Digitally signed purchases, with a certificate linked to credit card numbers

None of these were feasible

# The State of Computing

- This was pre-Windows 95—and Windows 3.1 did not come with a TCP/IP stack, though you could buy an add-on package from another vendor
- Home Internet use was exclusively by dial-up, with a maximum speed of 28.8K bps
- There was no such thing as e-commerce, so no one knew if it could work—Netscape was a startup
- Computers were relatively slow, DES and RC4 were the only encryption algorithms available in the US
- Public key cryptography in general, and RSA in particular, were covered by U.S. patents

# Packet-Layer Encryption

- A strong solution—can protect *all* traffic in or out of a machine
- (We'll discuss this in detail much later in the semester)
- Generally requires access to kernel or device driver code, though it can sometimes be done in other ways if interfaces are standard
- The real bottleneck: corporate servers, which ran vendor-supplied operating systems; vendors had no incentive to cooperate with a small, unknown start-up
- Conclusion: not feasible for Netscape

# Transport-Layer (TCP) Encryption

- At least as intrusive as network-layer encryption
- No standardized interfaces for third-party solutions
- Conclusion: not feasible for Netscape

# Digitally-Signed Credit Card Transactions

- In a strong sense, the best possible answer—it protected transactions, not traffic
- The actual items purchased could be part of the signed data
- Vendors would have something strong to show to a judge
- But—the users' certificates would have to be linked to their credit cards for vendors to be paid promptly
- Netscape couldn't do that; it wasn't a bank
- Banks didn't understand e-commerce
- There was no back-end infrastructure for merchants to use to collect payments
- Existing retailers understood and could process credit cards, not these weird digital signatures
- Conclusion: not feasible for Netscape

# SSL: Secure Socket Layer

- Solution: do encryption in the applications—the browser and the server—above TCP
- Use this to send credit card information to merchants
- Requires special code in these applications—but Netscape controlled them
- Operating system-independent; does not require deals with banks or OS vendors

Architecturally, it was the worst solution, with exactly one advantage: it was doable



- SSL 1.0 never saw the light of day
- Netscape released SSL 2.0 in 1995
- In 1996, they replaced it with SSL 3.0, to fix some security flaws
- In 1999, the IETF published a variant named TLS (Transport Layer Security) 1.0
- Development has continued; TLS 1.3 was released two years ago

# Design Principles

- General-purpose encryption
- Assume arbitrarily powerful enemies
- Must provide confidentiality and integrity
- Run over TCP; let it handle (benign) error correction and retransmission
- Flexibility to handle evolving needs and algorithms
- Adapt to the real world
- The result: the most important encryption mechanism on the Internet—but a very complex protocol

- Minimize number of round trips—communications are limited by the speed of light, which is finite!
  - The circumference of the Earth is 40,000 km
  - The speed of light in fiber is  $\frac{2}{3}c$
  - The *minimum* latency to reach a server halfway around the world is thus 100 ms., which is perceptible—and fiber isn't direct
- Public key operations are slow; minimize them (especially in 1994)
- HTTP is stateless—but if both sides can retain some encryption state, resuming a session can be much faster

# Statelessness

- Every HTTP transaction is independent—the connection between browser and server is closed after each download
- (No longer strictly true)
- Something is needed to link together different web views, for logins, preferences, shopping carts, etc.
- This was the original purpose for cookies—their use for tracking and advertising came later
- SSL had to live in that world

- Negotiate version of SSL (and later TLS)
- Algorithm agility; selection negotiated between the two ends
  - Accommodate newer algorithms
  - Permit different cost/security tradeoffs
  - US export rules
- Certificates from either side, both sides, or neither
- Forward secrecy (originally optional)

# Forward Secrecy

- Alice sends Bob a message encrypted to Bob's public key
- An attacker records it and later hacks Bob's system to steal his private key
- They can now read all old messages
- Forward secrecy is a way to prevent that

# Achieving Forward Secrecy

- Alice and Bob first do a Diffie-Hellman exchange:

$$A \rightarrow B : g^{r_A} \bmod p$$

$$B \rightarrow A : g^{r_B} \bmod p$$

- $g^{r_A r_B} \bmod p$  is now a shared secret
- Authenticate this exchange using digital signatures
- Once  $r_A$  and  $r_B$  are destroyed—and they're one-time use random numbers, not long-term keys—there is no way to reconstruct this exchange
- Old conversations are safe, even if Bob is later hacked


- SSL optionally performed compression. Why?



# Compression

- SSL optionally performed compression. Why?
- Remember the slow modem speeds—compression was very important for performance
- Modems, in fact, could do compression
- But you can't compress ciphertext! Why not?

# Compression

- SSL optionally performed compression. Why?
  - Remember the slow modem speeds—compression was very important for performance
  - Modems, in fact, could do compression
  - But you can't compress ciphertext! Why not?
  - Ciphertext is random, i.e., high entropy, and hence is not compressible
-  You have to compress before you encrypt
- N.B.: JPGs are also high entropy, but there were very few images transmitted back then—digital cameras were very rare and quite expensive

- Many protocols include version numbers, to permit evolution while maintaining backwards compatibility.
- It turns out to be surprisingly hard to get that right
- **Define the semantics of the version field in the very first release**
- This is especially tricky if you use major.minor versions or major.minor.update—but it's crucially important

# Downgrade Attacks

- Suppose that version 1.3 of a cryptographic protocol has a security risk, so you upgrade to 1.4
- But you need to support communication with sites that haven't upgraded, so you advertise both, 1.4 preferred, in your negotiation:  
 $A \rightarrow B : \{1.4, 1.3\}$
- Bob also supports both and should respond with  $\{1.4\}$
- But an attacker replaces your message with  $\{1.3\}$ , which Bob of course accepts
- Defending against this is tricky and protocol-dependent (and out of scope for this class); it often involves things like signatures over the proposed versions and the negotiated key

# Changes Since 1994

- Some of the changes are obvious: add newer ciphers, e.g., AES and elliptic curve, increase key lengths, delete insecure primitives (DES, MD5, RC4)
- Things like compression are no longer needed
- Adaptations to buggy implementations
- Support for *virtual hosts*: multiple websites at one IP address
- Make it harder for governments to spy

# Questions?



(Great blue heron, Central Park, February 16, 2019)