HTTPS: Attacks and Defense

1

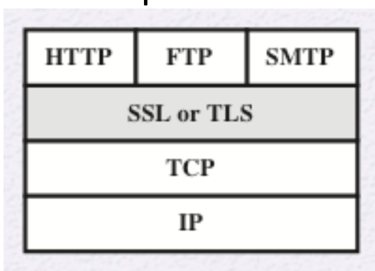
SSL

- **SSL** (Secure Socket Layer) -- Netscape
  - Version 2.0 -- Broken, don't use (disabled by default in modern browsers)
  - Version 3.0 (older but still in use)
- **TLS** (Transport Layer Security) -- IETF Standard
  - Version 1.0, 1.1, 1.2 (commonly used),
  - Version 1.3 (draft)

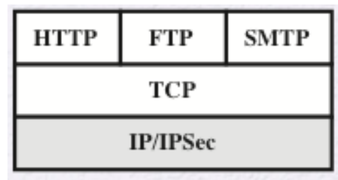
2

Different locations of security implementations

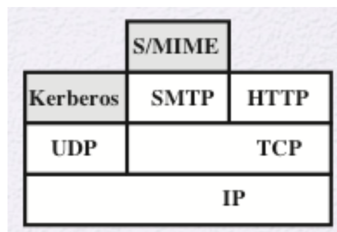
- Transport level



- Network level



- Application level



3

Threat model

- Controls infrastructure (routers, DNS, wireless access points)
- Passive attacker: only eavesdrops
- Active attacker: eavesdrops, injects, blocks, and modifies packets
- What examples?
  - Internet Cafe, hotel, CSE, fake web site,
- Does not protect against:
  - Intruder on server
  - Spyware on client
  - SQL injection, XSS, CSRF

4

Public-key cryptography

- Bob generates: SK_Bob, PK_Bob
- Alice can encrypt messages to Bob:
  - using PK_Bob encrypts messages, only Bob can decrypt
- Bob can sign messages that Alice can verify:
  - using SK_Bob signs message, anyone can verify
- What was the hard problem we haven't yet solved about this?

5

Certificates

- This public key with SHA-256 hash (XXX) belongs to the site (name, e.g., Amazon.com)
  - Signed by a trusted authority (digital signature)
- Your browsers (e.g., Firefox, Chrome) trust a specific set of CAs as root CAs
  - Shipped with the public keys of the root CAs
  - Why do we need more than 1?

6

Certificates

How does Alice (web browser) obtain PK_Bob?

Browser (Alice) | **Server (Bob)** | **Certificate Authority (CA)**
[Think of like a notary]

(Knows PK_CA) | (Keeps SK_CA Secret)

1. Choose (SK,PK)

--- PK and proof he is "Bob" -->

2. Checks proof

<-- Signs certificate with SK_CA ----
"Bob's key is PK -- Signed, CA"

3. Keeps cert on file

<-- Sends cert to Alice ----
"Bob's key is PK -- Signed, CA”

4. Verifies signature on cert.

7

Certificates verification

- How is identity verification done?
  – Typically 'DV' (domain verification) – just an email based challenge to the address in the domain registration records (Or some default email address); minimally secure.
- Cert has expiration date (e.g., one year ahead) [-- Why?]

8

SSL Certificates

- A trusted authority vouches that a certain public key belongs to a particular site
- Format called x.509 (complicated)
- Browsers ship with CA public keys for large number of trusted CAs [accreditation process]
- Important fields:
  - Common Name (CN) [e.g., *.google.com]
    Expiration Date [e.g. 2 years from now]
    Subject's Public Key
    Issuer -- e.g., Verisign
    Issuer's signature
- Common Name field
  - Explicit name, e.g. eecs.umich.edu
  - Or wildcard, e.g. *.umich.edu

9

Certificate Chains

- CA can delegate ability to generate certificates for certain names: Intermediate CAs
- Root CA signs "certificate issuing certificate" for delegated authority
- Delegated authority signs cert for "eecs.umich.edu"
  - Delegated CA certificate: "pubkey=.... is allowed to sign certs for *.umich.edu"
- Browser that trusts root can examine certs to establish validity -- "Chain of trust"
- How to find out about all the CAs?
- More than 1000 trusted parties today, can sign for any domain – huge problem!

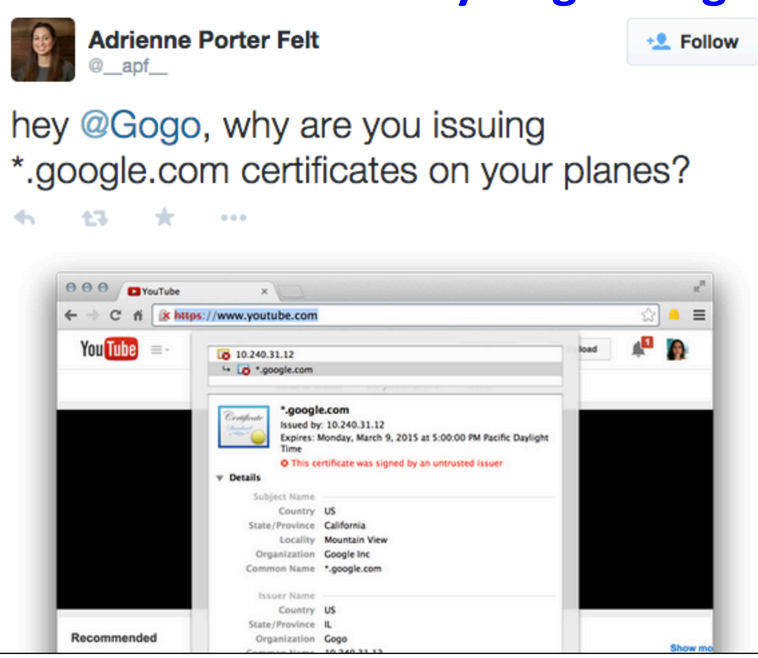
10

How to invalidate certificates?

- Expiration date of certs
- Certificate invalidation
- What happens if a CA's secret key is leaked?
  - Can we trust the old certs from that CA?
- Interesting fact:
  - Google has instrumented Chrome such that when it observes a certificate for Google.com that it doesn't recognize, it panics…. (has happened several times)

11

Man in the middle by Gogo inflight



12

Self-signed Certificates

- Issuer signs their own certificate
  - A loop in the owner and signer
- Avoid CA fees, useful for testing
- Browsers display warnings that users have to override
- Protects only against passive attacker

13

Question

- Can a web sever obtain SSL server certificates from two or more certification authorities? Justify your answer.
- Yes, the administrator of the website can apply for a certificate from multiple certification authorities. However, only one of these certificates is returned to the client as part of the HTTPS protocol.

14

Exercise

- Which of the following security goals are addressed by the HTTPS protocol: (a) privacy, (b) confidentiality, (c) availability, (d) integrity,
- Does HTTPS authenticate the user?
  - User names and passwords are often used.

15

How do we translate?

Cryptographic Primitives

Symmetric Encryption
RSA
PKI
HMAC
Certificate
Public Key
RC4
Diffie-Hellman
DSA
ECDSA
Asymmetric Encryption

16

How do we translate?

Cryptographic Primitives

Symmetric Encryption
RSA
PKI
HMAC
Certificate
Public Key
RC4
Diffie-Hellman
DSA
ECDSA
Asymmetric Encryption

Objectives

Message Integrity

Confidentiality

Authentication

17

How do we translate?

Cryptographic Primitives

Symmetric Encryption
RSA
PKI
HMAC
Certificate
Public Key
RC4
Diffie-Hellman
DSA
ECDSA
Asymmetric Encryption

Typical HTTPS Connection

18

Case Study: SSL/TLS

- Arguably the most important (and widely used) cryptographic protocol on the Internet
- Almost all encrypted protocols (minus SSH) use SSL/TLS for transport encryption
- HTTPS, POP3, IMAP, SMTP, FTP, NNTP, XMPP (Jabber), OpenVPN, SIP (VoIP), …

19

SSL vs. TLS

- SSL := Secure Sockets Layer (Netscape)
- TLS := Transport Layer Security (IETF)
- Terms are used interchangeably
- SSL 3.0 is predecessor to TLS 1.0

20

Browser TLS Support

Browser	Platforms	TLS 1.0	TLS 1.1	TLS 1.2
Chrome 0–22	Linux, Mac OS X, Windows (XP, Vista, 7)[a]	Yes	No	No
Chrome 22–	Linux, Mac OS X, Windows (XP, Vista, 7)[a]	Yes	Yes	No
Firefox 2–	Linux, Mac OS X, Windows (XP, Vista, 7)	Yes[34]	No[35]	No[36]
IE 1–7	Mac OS X, Windows (XP, Vista, 7)[b]	Yes	No	No
IE 8–	Windows 7[b]	Yes	Yes	Yes
Opera 10–	Linux, Mac OS X, Microsoft Windows[c]	Yes	Yes, disabled	Yes, disabled
Safari 5–	Mac OS X, Windows (XP, Vista, 7)[d]	Yes	?	?

source: http://en.wikipedia.org/wiki/Transport_Layer_Security

21

Where does TLS live?

Application (HTTP)

Transport (TCP)

Network (IP)

Data-Link (1gigE)

Physical (copper)

22

Goals

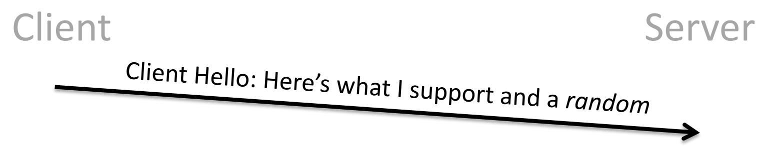
- ✓ Confidentiality (Symmetric Crypto)
- ✓ Message Integrity (HMACs)
- ✓ Authentication (Public Key Crypto)

23

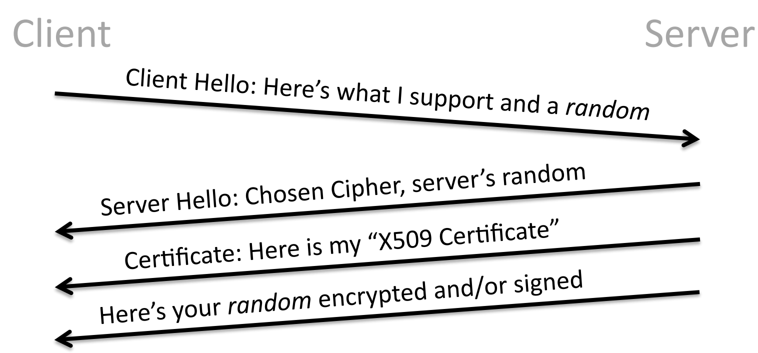
Client Server

"the handshake"

24

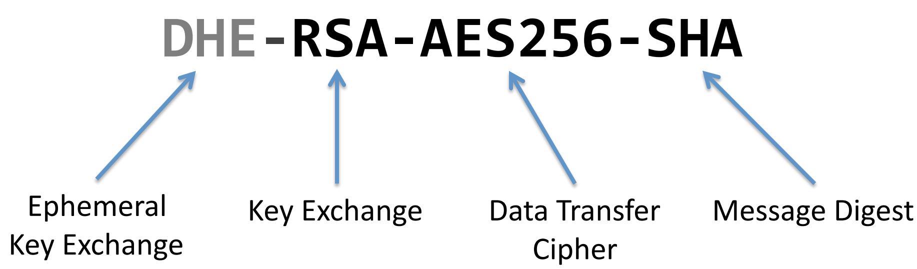

Client
Server
Client Hello: Here's what I support and a *random*

25

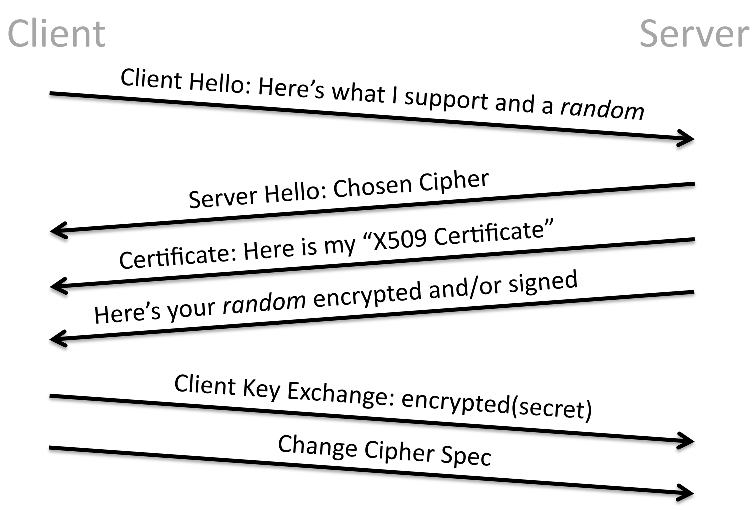

Client
Server
Client Hello: Here's what I support and a *random*
Server Hello: Chosen Cipher, server's random
Certificate: Here is my "X509 Certificate"
Here's your *random* encrypted and/or signed

26

Cipher Suites



27



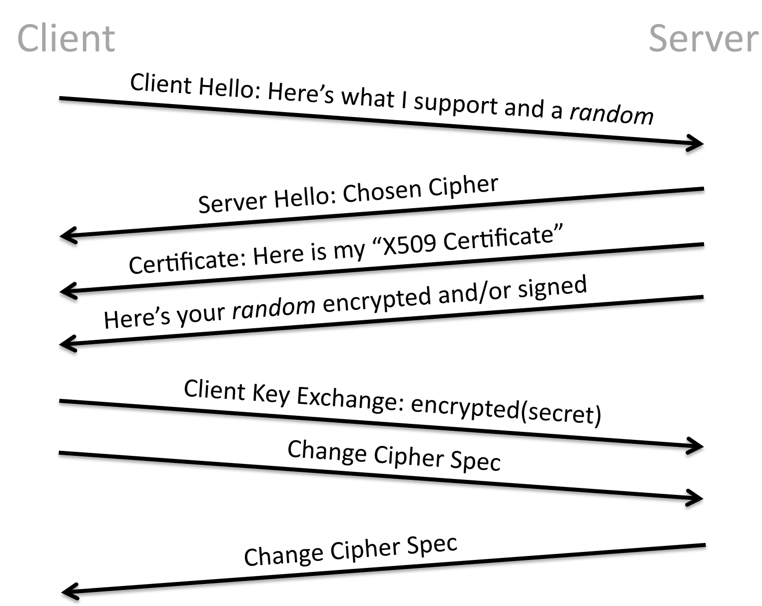
28

HTTPS key exchange

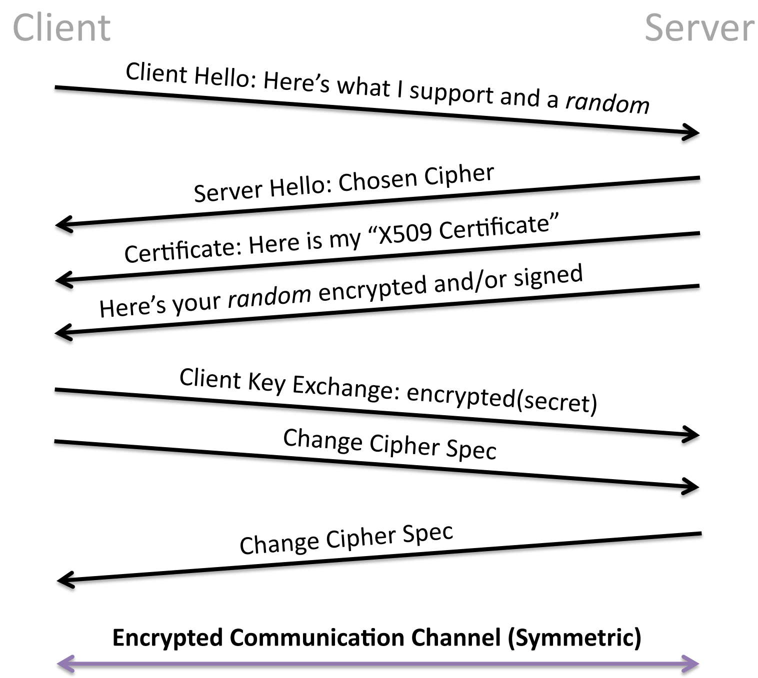
1. RSA key exchange
   – Use RSA for encryption to achieve confidentiality
2. Ephemeral Diffie Hellman (EPH)
   – Use RSA for signature to achieve authentication

- Which one to use?
  – RSA is simpler, EPH is more work
  – What about forward secrecy guarantees?
- At the end of the exchange, a secret is used to generate 4 keys

29



30



31

X509 Certificates

Subject: C=US/O=Google Inc/CN=www.google.com
Issuer: C=US/O=Google Inc/CN=Google Internet Authority
Serial Number: 01:b1:04:17:be:22:48:b4:8e:1e:8b:a0:73:c9:ac:83
Expiration Period: Jul 12 2010 - Jul 19 2012
Public Key Algorithm: rsaEncryption
Public Key: 43:1d:53:2e:09:ef:dc:50:54:0a:fb:9a:f0:fa:14:58:ad:a0:81:b0:3d 7c:be:b1:82:19:b9:7c3:8:04:e9:1e5d:b5:80:af:d4:a0:81:b0:b0:68:5b:a4:a4 :ff:b5:8a:3a:a2:29:e2:6c:7c3:8:04:e9:1e5d:b5:7c3:8:04:e9:39:23:46

Signature Algorithm: sha1WithRSAEncryption

Signature: 39:10:83:2e:09:ef:ac:50:04:0a:fb:9a:f0:fa:14:58:ad:a0:81:b0:3d 7c:be:b1:82:19:b9:7c3:8:04:e9:1e5d:b5:80:af:d4:a0:81:b0:b0:68:5b:a4:a4 :ff:b5:8a:3a:a2:29:e2:6c:7c3:8:04:e9:1e5d:b5:7c3:8:04:e9:1e:5d:b5

32

Certificate Chains

Mozilla Firefox Browser

Trust everything signed by this "root" certificate

Subject: C=US/…/OU=Equifax Secure Certificate Authority
Issuer: C=US/…/OU=Equifax Secure Certificate Authority
Public Key:
Signature: 39:10:83:2e:09:ef:ac:50:04:0a:fb:9a:38:c9:d1

I authorize and trust this certificate; here is my signature

Subject: C=US/…/CN=Google Internet Authority
Issuer: C=US/…/OU=Equifax Secure Certificate Authority
Public Key:
Signature: be:b1:82:19:b9:7c:5d:28:04:e9:1e:5d:39:cd

I authorize and trust this certificate; here is my signature

Subject: C=US/…/O=Google Inc/CN=*.google.com
Issuer: C=US/…/CN=Google Internet Authority
Public Key:
Signature: bf:dd:e8:46:b5:a8:5d:28:04:38:4f:ea:5d:49:ca

33

Certificate Authority Ecosystem

Each browser trusts a set of CAs

- CAs can sign certificates for new CAs
- CAs can sign certificates for any web site

If a single CA is compromised, then the entire system is compromised

We ultimately place our complete trust of the Internet in the weakest CA

34

Immediate Concerns

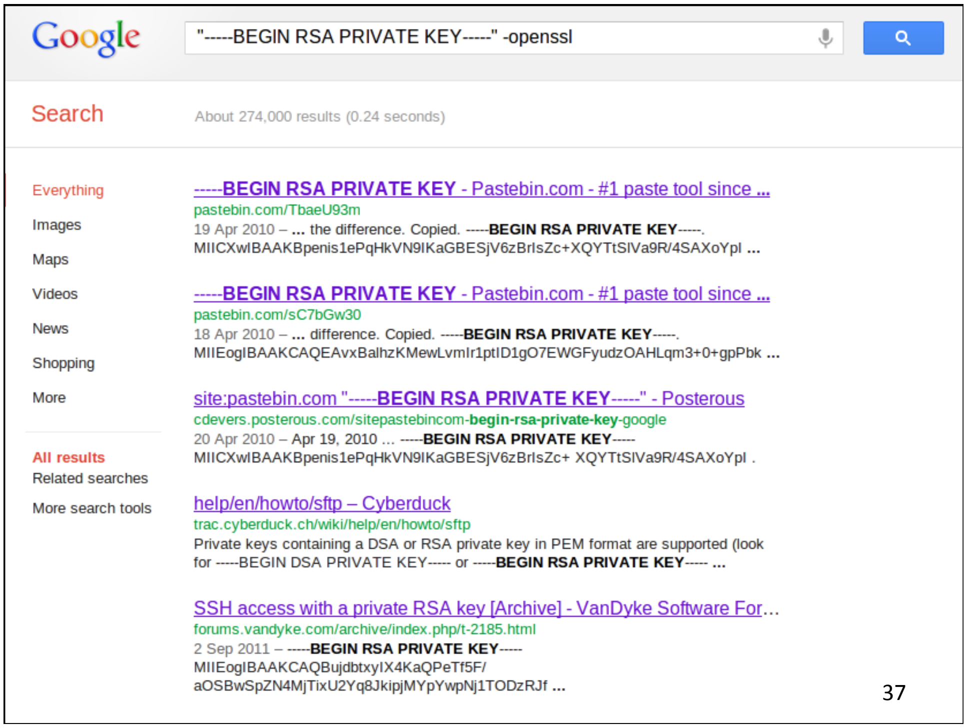
- Nobody has any idea who these CAs are…
- 1,733 *umich*-known browser trusted CAs
- History of CAs being hacked (e.g. Diginotar)
- Oooops, Korea gave every elementary school, library, and agency a CA certificate (1,324)
  – Luckily invalid due to a higher-up constraint

35

Getting a Certificate

- Certificates are free (from StartSSL)!
  – https://cert.startcom.org/
- Certificates are cheap if you don't want to use the impossible-to-use StartSSL web interface
- Identity is almost always validated via e-mail to the default e-mail addresses
- Setting up SSL is hard. People are terrible at it.

36



Attack Vectors

- Attack the weakest Certificate Authority
- Attack browser implementations
- Magically notice a bug in a key generation library that leads you to discovering all the private keys on the Internet
- Attack the cryptographic primitives
  - Math is hard, let's go shopping!

38

SSL in the browser

- Lock icon
  - HTTPS cert must be issued by a CA trusted by browser
  - HTTPS cert is valid (e.g., not expired or revoked)
  - CommonName in cert matches domain in URL
- Extended Validation (EV) certificates
  - CA does extra work to verify identity -- expensive, but more secure
- Invalid certificate warnings

39

Attacking site design

- SSLstrip attack
  - Proxy through the content w/o HTTPS
- Defense
  - Default HTTPS for all web sites?
  - HSTS (hypertext strict transport security): header says: always expect HTTPS, enforced by browsers.
  - HTTPS Everywhere: browser extension
  - EV: Extended Validation (compared to DV: Domain Validation)

40

Attacking site design

- Mixed Content attack -- Page loads over HTTPS but contains content over HTTP
  - e.g. JavaScript, Flash
  - Active attacker can tamper with HTTP content to hijack session
- Defense: Browser warnings: ["This page contains insecure content"],
  - but inconsistent and often ignored

41

UI interface based attacks

- Invalid certs
  - Expired, Common Name != URL, unknown CA (e.g., self-signed)
- Defense: browser warnings, anti-usability to bypass…
- Picture-in-picture attack: spoof the user interface
  - Attacker page draws fake browser window with lock icon
- Defense: individualized image

42

Attacking the PKI: CA compromise Example: DigiNotar

- DigiNotar ***was*** a Dutch Certificate Authority
- On June 10, 2011, **`*.google.com`** cert was issued to an attacker and subsequently used to orchestrate MITM attacks in Iran
- Nobody noticed the attack until someone found the certificate in the wild… and posted to *pastebin*

43

DigiNotar Contd.

- DigiNotar later admitted that dozens of fraudulent certificates were created
- Google, Microsoft, Apple and Mozilla all revoked the root Diginotar certificate
- Dutch Government took over Diginotar
- Diginotar went bankrupt and died

44

Attacking the PKI: Hash collisions

- MD5 is known to be broken -- Can generate collisions
- In 2008, researchers showed that they could create a rogue CA certificate using an MD5 collision
- Attack: Make colliding messages A, B, with same MD5 hash:
  - A: Site certificate: "cn=attack.com, pubkey=...."
  - B: Delegated CA certificate: "pubkey=.... is allowed to sign certs for *"
  - Get CA to sign A -- Signature is Sign(MD5(message))
  - Signature also valid for B (same hash)
  - Attacker is now a CA!
  - Make a cert for any site, browsers will accept it

45

MD5 considered harmful

- MD5 CA certificates still exist, but CAs have stopped signing certificates with them
  - 879,705 certificates still have MD5 signatures

46

Attacking implementations: Null Termination Attack

- ASN.1 utilizes Pascal-style strings
- Web browsers utilize use C-style strings
- Announced by Moxie in 2009

`gmail.com\0.badguy.com`

47

Null Termination Attack

- www.attacker.com
  - [CAs verify cert by looking up who owns the last part of the domain via DNS record]
  - emails "webmaster@attack.com" --> "Click here to validate cert request"
- x.509 certs encode CN field as a Pascal string (length +data)
- Browsers copy it into a C string (data+\0)
- What if CA contains "\0"?
  - www.paypal.com\0.attacker.com?
  - CA contacts "attacker.com" to verify (last part of domain name)
  - Browsers copy to C string, terminates at "\0" -- see only paypal.com
  - Attacker now has a cert that works for Paypal!

48

Other implementation-based attacks

- Goto fail, Feb. 2014 (Apple SSL bug; skipped certificate check for almost a year!)
- Heartbleed, April 2014 (OpenSSL bug; leaked data, possibly including private key!)
- Mozilla BERserk vulnerability, Oct 2014 (Bug in verifying cert signatures, allowed spoofing certs, probably since the beginning….!)

49

Take aways

- Use HTTPS! It's so much better than nothing .
- SSL keeps breaking. Use it, but don't rely on it exclusively.
- Have a backup plan for times when it's broken.

50

Exercise

- Suppose a web client and web server for a popular shopping web site have performed a key exchange so that they now share a secret session key. Describe a secure method for the client to visit the shopping site.
- Your solution cannot use HTTPS, so it does not need to achieve confidentiality. It should be resistant to HTTP session hijacking even from someone who can sniff all the packets.

- **Solution:** Seed the PRNG with the secret key and include in each HTTP request the next pseudo-random number in the sequence, as well as a userID, as a part of the URL. The server can determine that this is the specified user, because even an eavesdropper would not be able to determine the next number in the PRNG.

51