Announcements

• Homework 2 due in 1 week
• Lab 2 (web security) out likely next week
**Problem:** How does Alice know that the public key she received is really Bob’s public key?
Review: Distribution of Public Keys

• Public announcement or public directory
  – Risks: forgery and tampering

• Public-key certificate
  – Signed statement specifying the key and identity
    • $\text{sig}_{\text{CA}}(\text{“Bob”}, \text{PK}_B)$

• Common approach: certificate authority (CA)
  – Single agency responsible for certifying public keys
  – After generating a private/public key pair, user proves his identity and knowledge of the private key to obtain CA’s certificate for the public key (offline)
  – Every computer is pre-configured with CA’s public key
You encounter this every day...

SSL/TLS: Encryption & authentication for connections
SSL/TLS High Level

• SSL/TLS consists of two protocols
  – Familiar pattern for key exchange protocols
• Handshake protocol
  – Use public-key cryptography to establish a shared secret key between the client and the server
• Record protocol
  – Use the secret symmetric key established in the handshake protocol to protect communication between the client and the server
Example of a Certificate
Hierarchical Approach

- Single CA certifying every public key is impractical
- Instead, use a trusted root authority (e.g., Verisign)
  - Everybody must know the root’s public key
  - Instead of single cert, use a certificate chain
    - \( \text{sig}_{\text{Verisign}}(\text{“AnotherCA”}, \text{PK}_{\text{AnotherCA}}), \text{sig}_{\text{AnotherCA}}(\text{“Alice”}, \text{PK}_A) \)
  - Not shown in figure but important:
    - Signed as part of each cert is whether party is a CA or not
- What happens if root authority is ever compromised?
Trusted(?) Certificate Authorities

![Keychain Access screenshot]

<table>
<thead>
<tr>
<th>Name</th>
<th>Kind</th>
<th>Expires</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdminCA-CD-T01</td>
<td>certificate</td>
<td>Jan 25, 2016, 4:36:19 AM</td>
</tr>
<tr>
<td>AffirmTrust Commercial</td>
<td>certificate</td>
<td>Dec 31, 2030, 6:06:08 AM</td>
</tr>
<tr>
<td>AffirmTrust Networking</td>
<td>certificate</td>
<td>Dec 31, 2030, 6:08:24 AM</td>
</tr>
<tr>
<td>AffirmTrust Premium</td>
<td>certificate</td>
<td>Dec 31, 2040, 6:10:36 AM</td>
</tr>
<tr>
<td>AffirmTrust Premium ECC</td>
<td>certificate</td>
<td>Dec 31, 2040, 6:20:24 AM</td>
</tr>
<tr>
<td>America Onli...ation Authority 1</td>
<td>certificate</td>
<td>Nov 19, 2037, 12:43:00 PM</td>
</tr>
<tr>
<td>America Onli...ation Authority 2</td>
<td>certificate</td>
<td>Sep 29, 2037, 7:06:00 AM</td>
</tr>
<tr>
<td>Apple Root CA</td>
<td>certificate</td>
<td>Feb 9, 2035, 1:40:36 PM</td>
</tr>
<tr>
<td>Apple Root CA - G2</td>
<td>certificate</td>
<td>Apr 30, 2039, 11:10:09 AM</td>
</tr>
<tr>
<td>Apple Root CA - G3</td>
<td>certificate</td>
<td>Apr 30, 2039, 11:19:06 AM</td>
</tr>
<tr>
<td>Apple Root Certificate Authority</td>
<td>certificate</td>
<td>Feb 9, 2025, 4:18:14 PM</td>
</tr>
<tr>
<td>Application CA G2</td>
<td>certificate</td>
<td>Mar 31, 2016, 7:59:59 AM</td>
</tr>
<tr>
<td>ApplicationCA</td>
<td>certificate</td>
<td>Dec 12, 2017, 7:00:00 AM</td>
</tr>
</tbody>
</table>
Turtles All The Way Down...

The saying holds that the world is supported by a chain of increasingly large turtles. Beneath each turtle is yet another: it is "turtles all the way down".

[Image from Wikipedia]
Many Challenges...

• Hash collisions
• Weak security at CAs
  – Allows attackers to issue rogue certificates
• Users don’t notice when attacks happen
  – We’ll talk more about this later in the course
• How do you revoke certificates?
Colliding Certificates

- **serial number**
- **validity period**
- **real cert domain name**
- **real cert RSA key**
- **X.509 extensions**
- **signature**

<table>
<thead>
<tr>
<th>set by the CA</th>
<th>chosen prefix (difference)</th>
<th>Hash to the same MD5 value!</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>real cert</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>domain name</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>RSA key</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>X.509 extensions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>signature</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>rogue cert</th>
<th>collision bits (computed)</th>
<th>identical bytes (copied from real cert)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>domain name</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>RSA key</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>X.509 extensions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>signature</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Valid for both certificates!

[Sotirov et al. “Rogue Certificates”]
Attacking CAs

Security of DigiNotar servers:
• All core certificate servers controlled by a single admin password (Pr0d@dm1n)
• Software on public-facing servers out of date, unpatched
• No anti-virus (could have detected attack)
Consequences

• Attacker needs to first divert users to an attacker-controlled site instead of Google, Yahoo, Skype, but then...
  – For example, use DNS to poison the mapping of mail.yahoo.com to an IP address
• “authenticate” as the real site
• ... decrypt all data sent by users
  – Email, phone conversations, Web browsing
More Rogue Certs

• In Jan 2013, a rogue *.google.com certificate was issued by an intermediate CA that gained its authority from the Turkish root CA TurkTrust
  – TurkTrust accidentally issued intermediate CA certs to customers who requested regular certificates
  – Ankara transit authority used its certificate to issue a fake *.google.com certificate in order to filter SSL traffic from its network

• This rogue *.google.com certificate was trusted by every browser in the world

• There are plenty more stories like this...
Certificate Revocation

• Revocation is **very** important

• Many valid reasons to revoke a certificate
  – Private key corresponding to the certified public key has been compromised
  – User stopped paying their certification fee to this CA and CA no longer wishes to certify them
  – CA’s private key has been compromised!

• Expiration is a form of revocation, too
  – Many deployed systems don’t bother with revocation
  – Re-issuance of certificates is a big revenue source for certificate authorities
Certificate Revocation Mechanisms

• Certificate revocation list (CRL)
  – CA periodically issues a signed list of revoked certificates
    • Credit card companies used to issue thick books of canceled credit card numbers
  – Can issue a “delta CRL” containing only updates

• Online revocation service
  – When a certificate is presented, recipient goes to a special online service to verify whether it is still valid
    • Like a merchant dialing up the credit card processor
Attempt to Fix CA Problems:

Certificate Transparency

- **Problem:** browsers will think nothing is wrong with a rogue certificate until revoked
- **Goal:** make it impossible for a CA to issue a bad certificate for a domain *without the owner of that domain knowing*
- **Approach:** auditable certificate logs
  - Certificates published in public logs
  - Public logs checked for unexpected certificates

[www.certificate-transparency.org](http://www.certificate-transparency.org)
Attempt to Fix CA Problems:

Certificate Pinning

• **Trust on first access:** tells browser how to act on subsequent connections

• HPKP – HTTP Public Key Pinning
  – Use these keys!
  – HTTP response header field “Public-Key-Pins”

• HSTS – HTTP Strict Transport Security
  – Only access server via HTTPS
  – HTTP response header field "Strict-Transport-Security"
Big Picture: Browser and Network

Browser

OS

Hardware

Network

request

reply

website
Where Does the Attacker Live?

Mitigation: SSL/TLS (not covered further)

Mitigation: Browser security model + web app security (next week)
Two Sides of Web Security

(1) Web browser
   – Responsible for securely confining content presented by visited websites

(2) Web applications
   – Online merchants, banks, blogs, Google Apps …
   – Mix of server-side and client-side code
     • Server-side code written in PHP, JavaScript, C++ etc.
     • Client-side code written in JavaScript (… sort of)
   – Many potential bugs: XSS, XSRF, SQL injection
But at least 3 actors!

User
+
Browser

A.com

Network

B.com
Browser: All of These Should Be Safe

• Safe to visit an evil website

• Safe to visit two pages
  – Simultaneously
  – Sequentially

• Safe delegation
Browser Security Model

**Goal 1:** Protect local system from web attacker
- Browser Sandbox

**Goal 2:** Protect/isolate web content from other web content
- Same Origin Policy