CSE 484 / CSE M 584: Computer Security and Privacy

SSL/TLS

Fall 2016

Ada (Adam) Lerner
lerner@cs.washington.edu

Thanks to Franzi Roesner, Dan Boneh, Dieter Gollmann, Dan Halperin, Yoshi Kohno, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...

We have all the pieces!

- Symmetric Encryption (privacy!)
- MACs (integrity!)
- Asymmetric Crypto (bootstrapping!)
- Certificate Authorities (authenticity!)

SSL/TLS

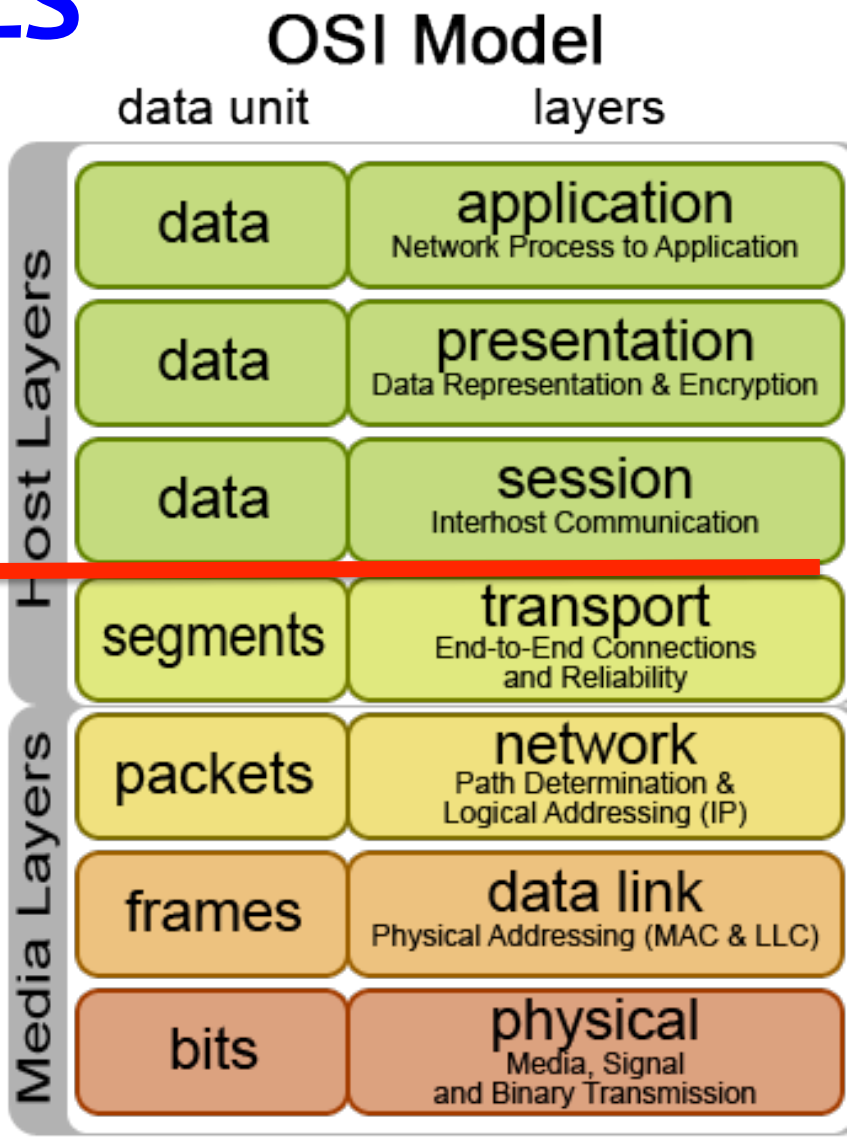
- Secure Sockets Layer and Transport Layer Security
  - Same protocol, new version (TLS is current)
- De facto standard for Internet security
  - "The primary goal of the TLS protocol is to provide privacy and data integrity between two communicating applications"
- Deployed in every Web browser; also VoIP, payment systems, distributed systems, etc.

SSL/TLS

- TLS is typically used on top of a TCP connection

TLS

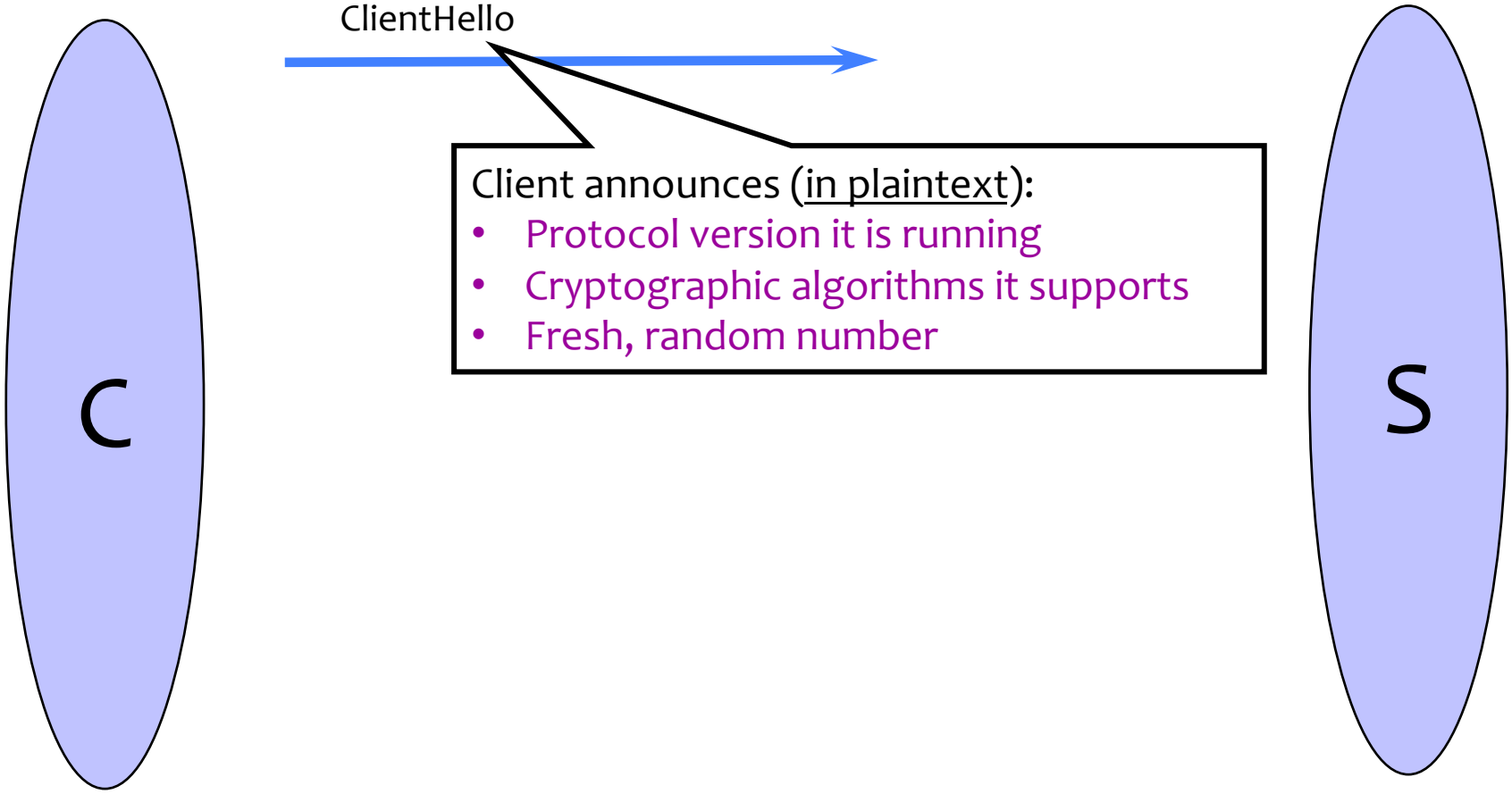
- Can be used over other transport protocols



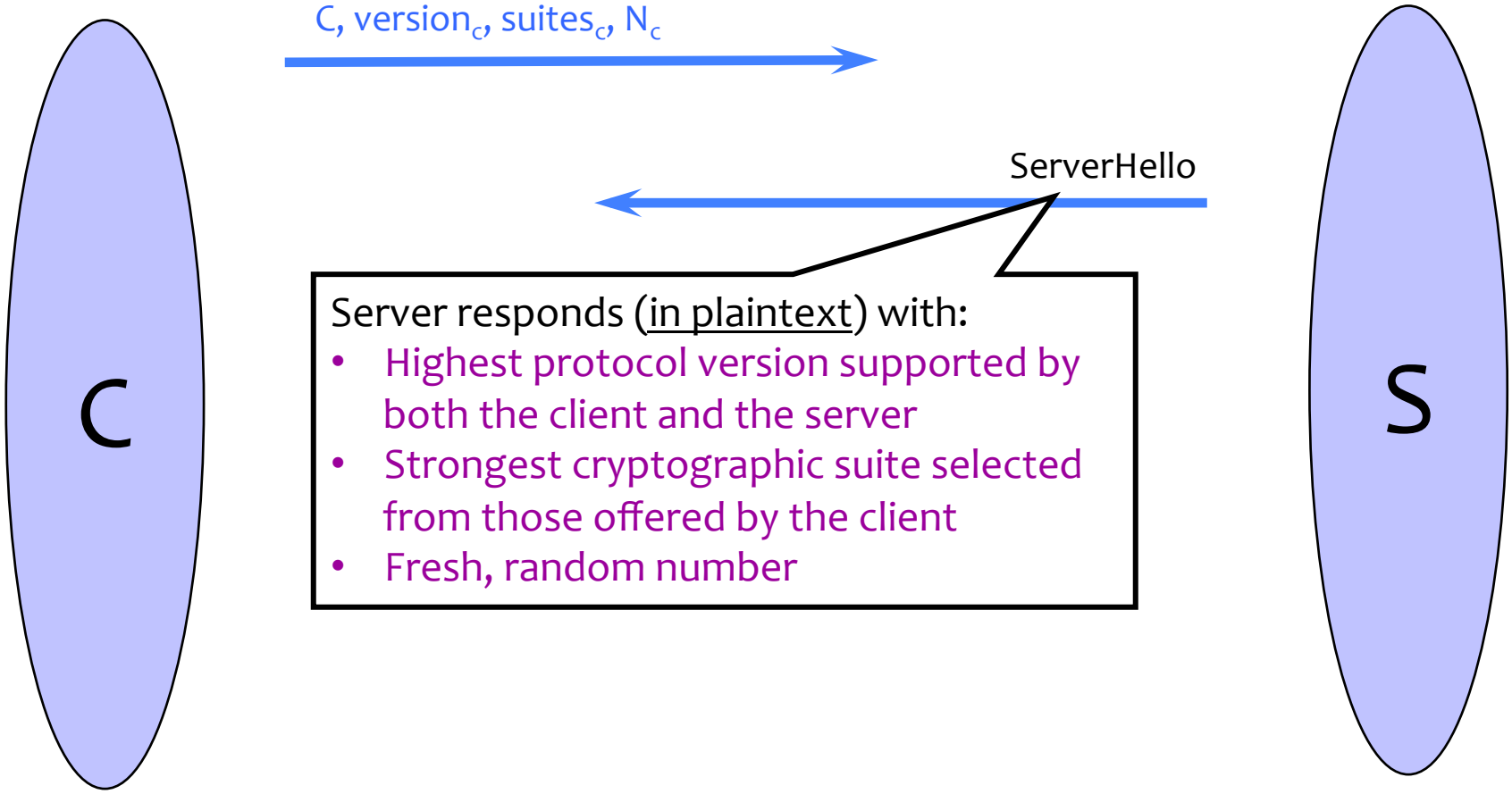
TLS Basics

- 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

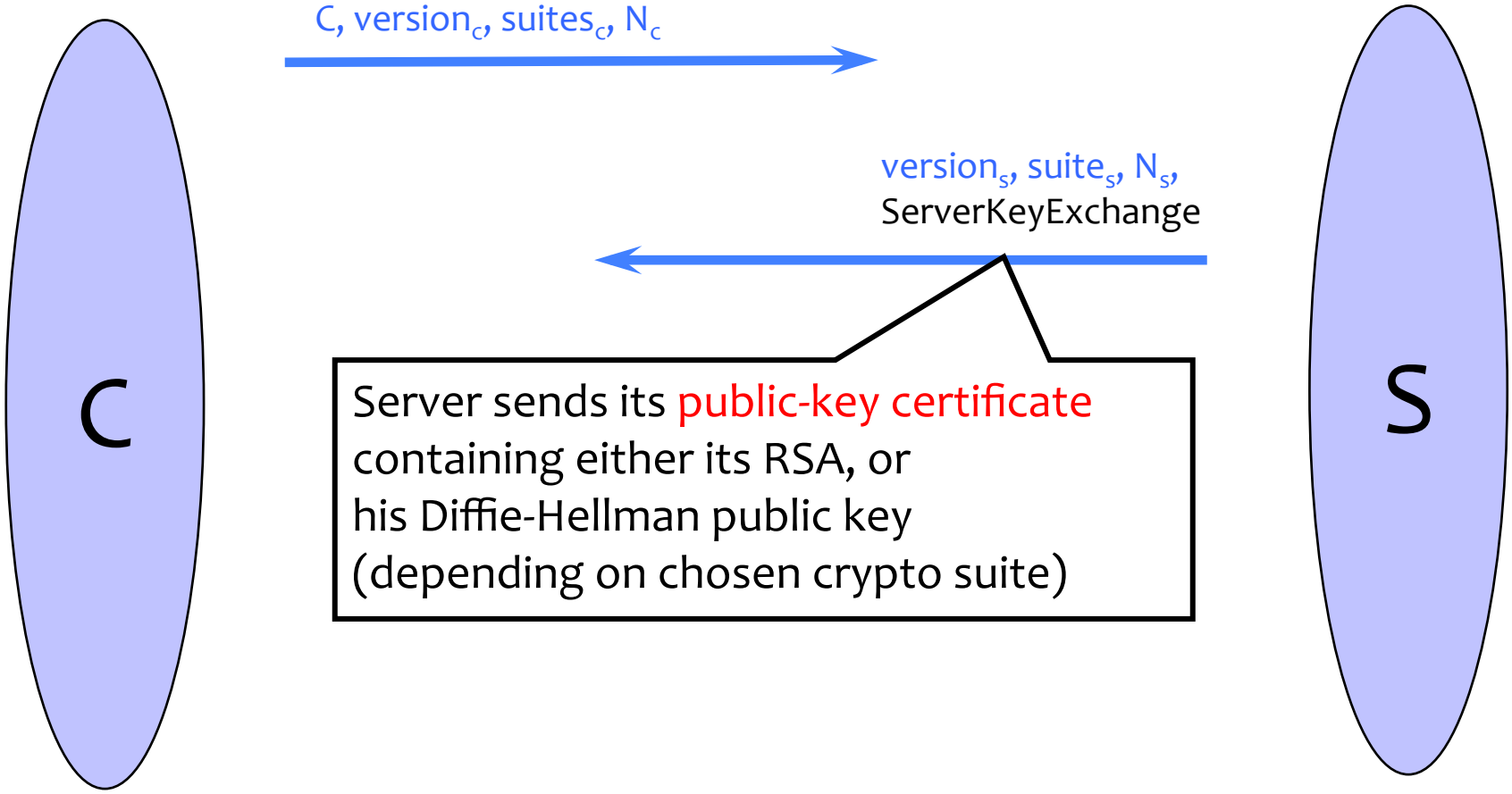
Basic Handshake Protocol



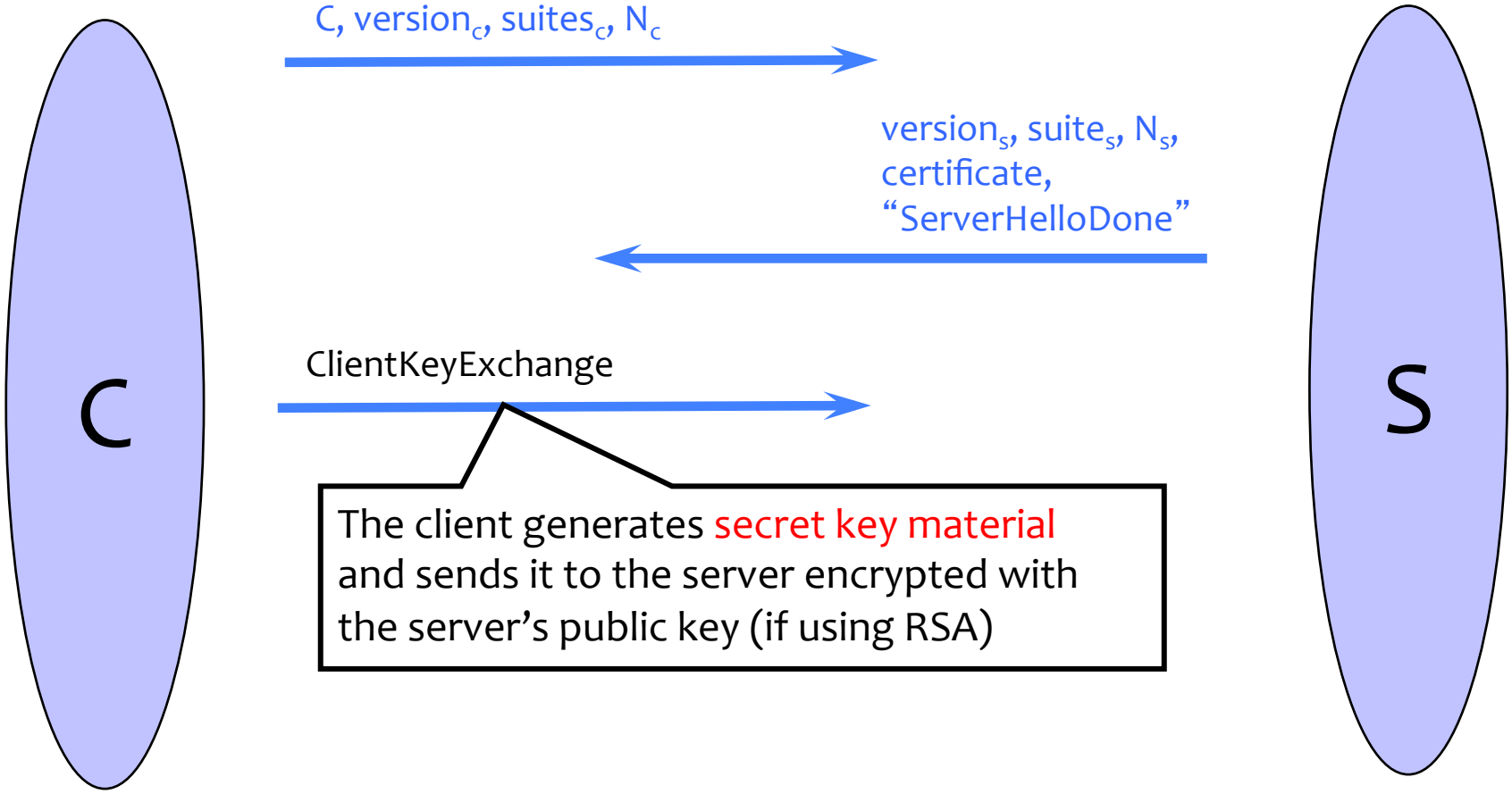
Basic Handshake Protocol



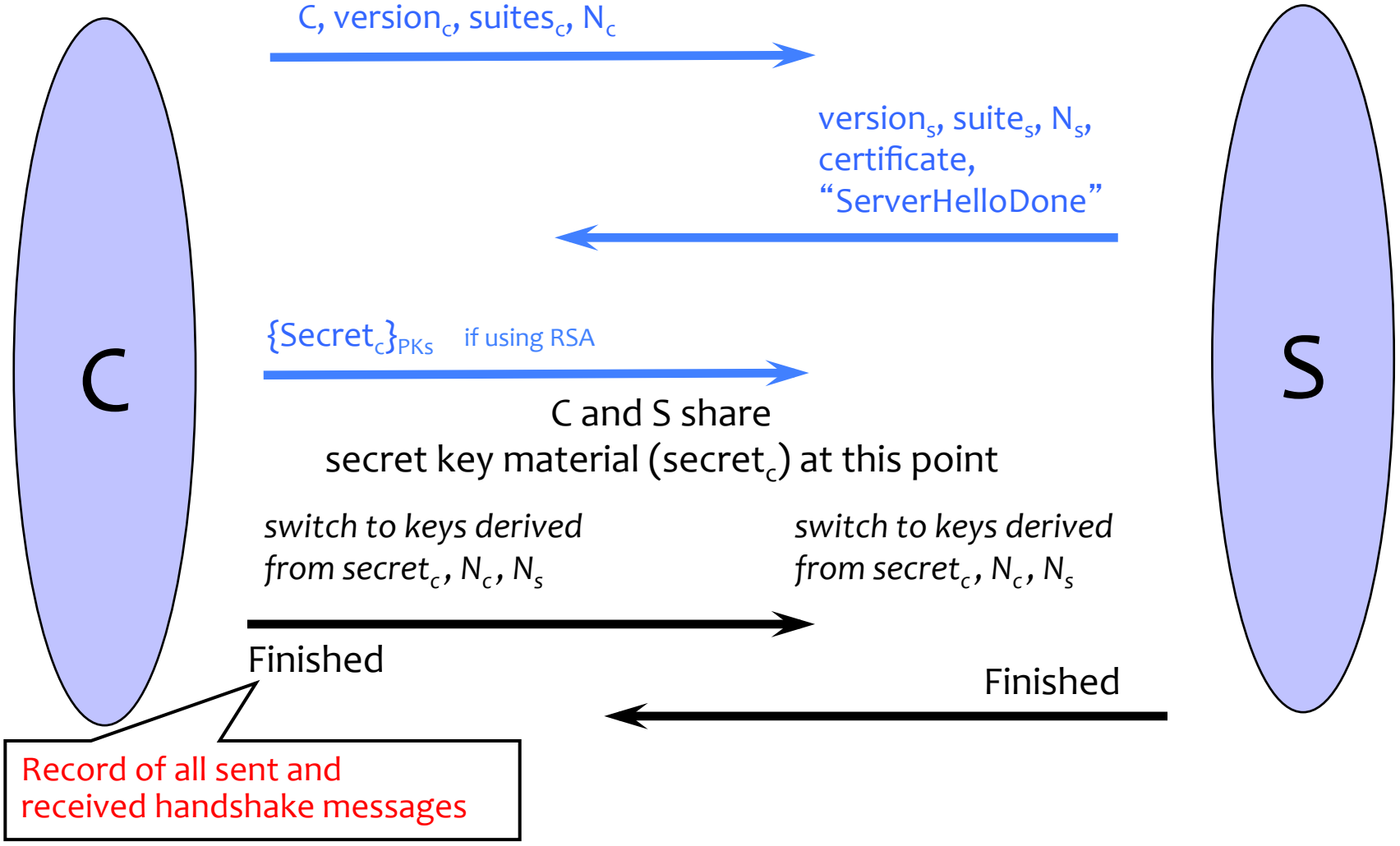
Basic Handshake Protocol



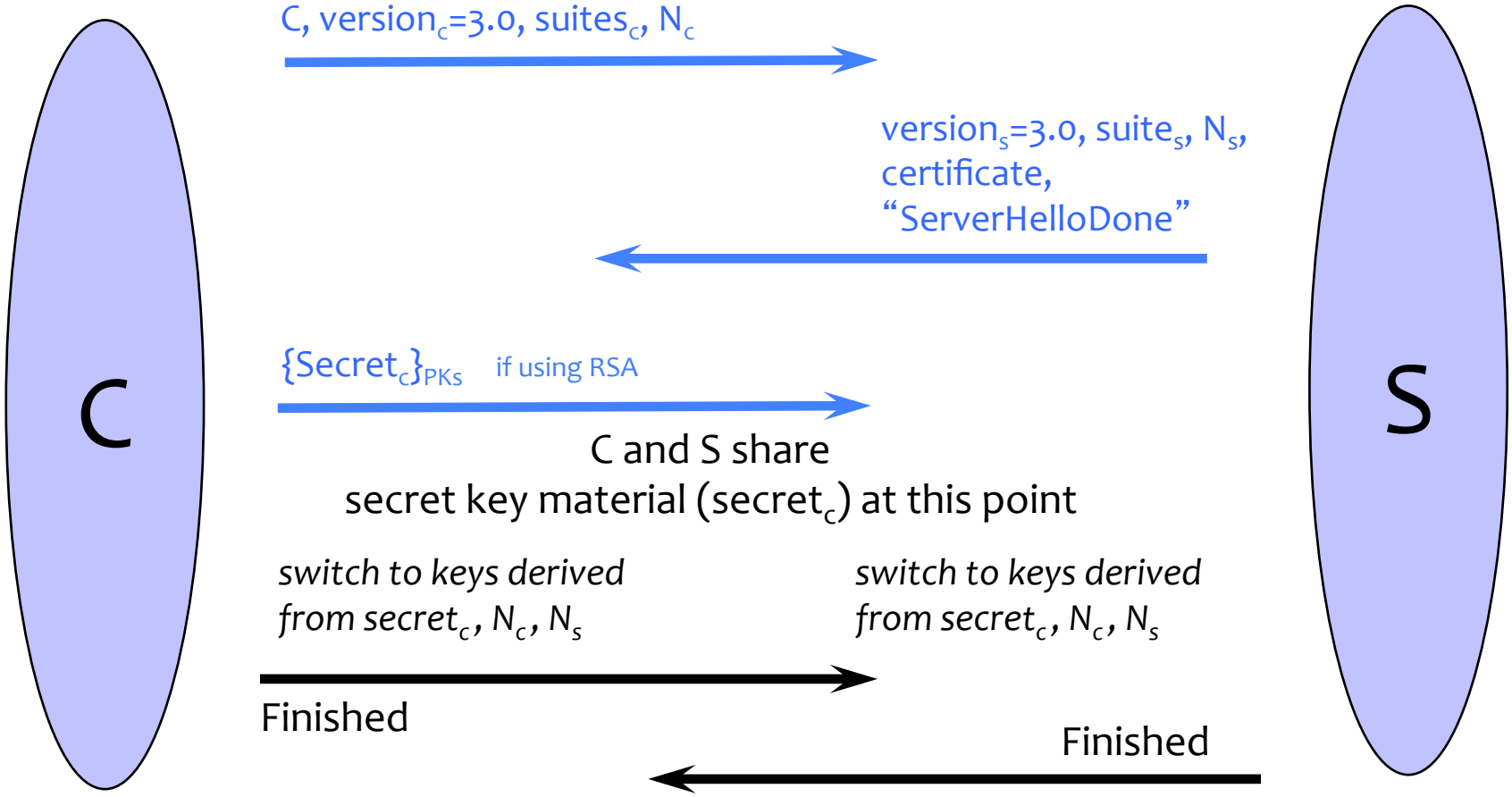
Basic Handshake Protocol



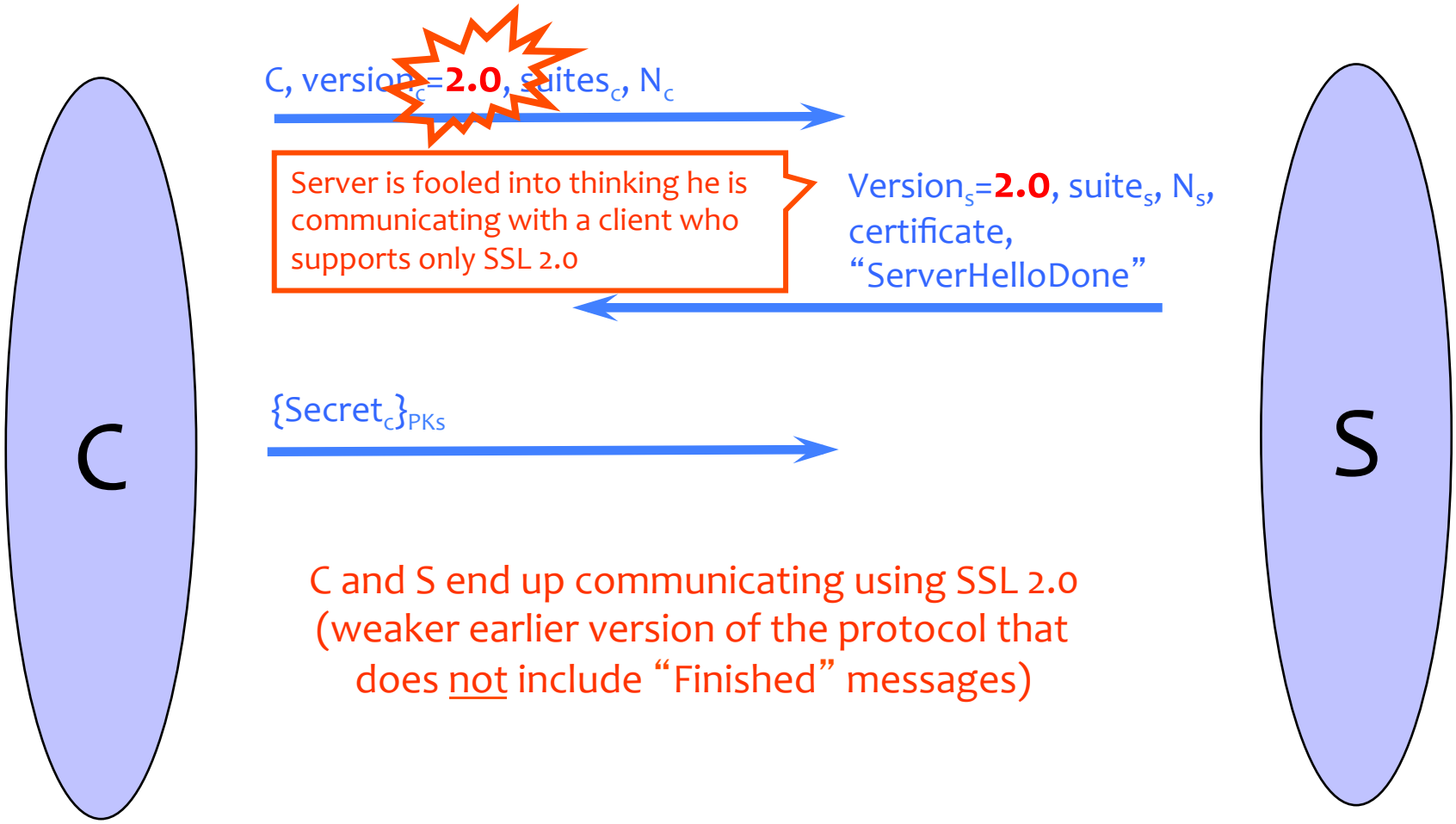
Basic Handshake Protocol



"Core" SSL 3.0 Handshake (Not TLS)



Version Rollback Attack

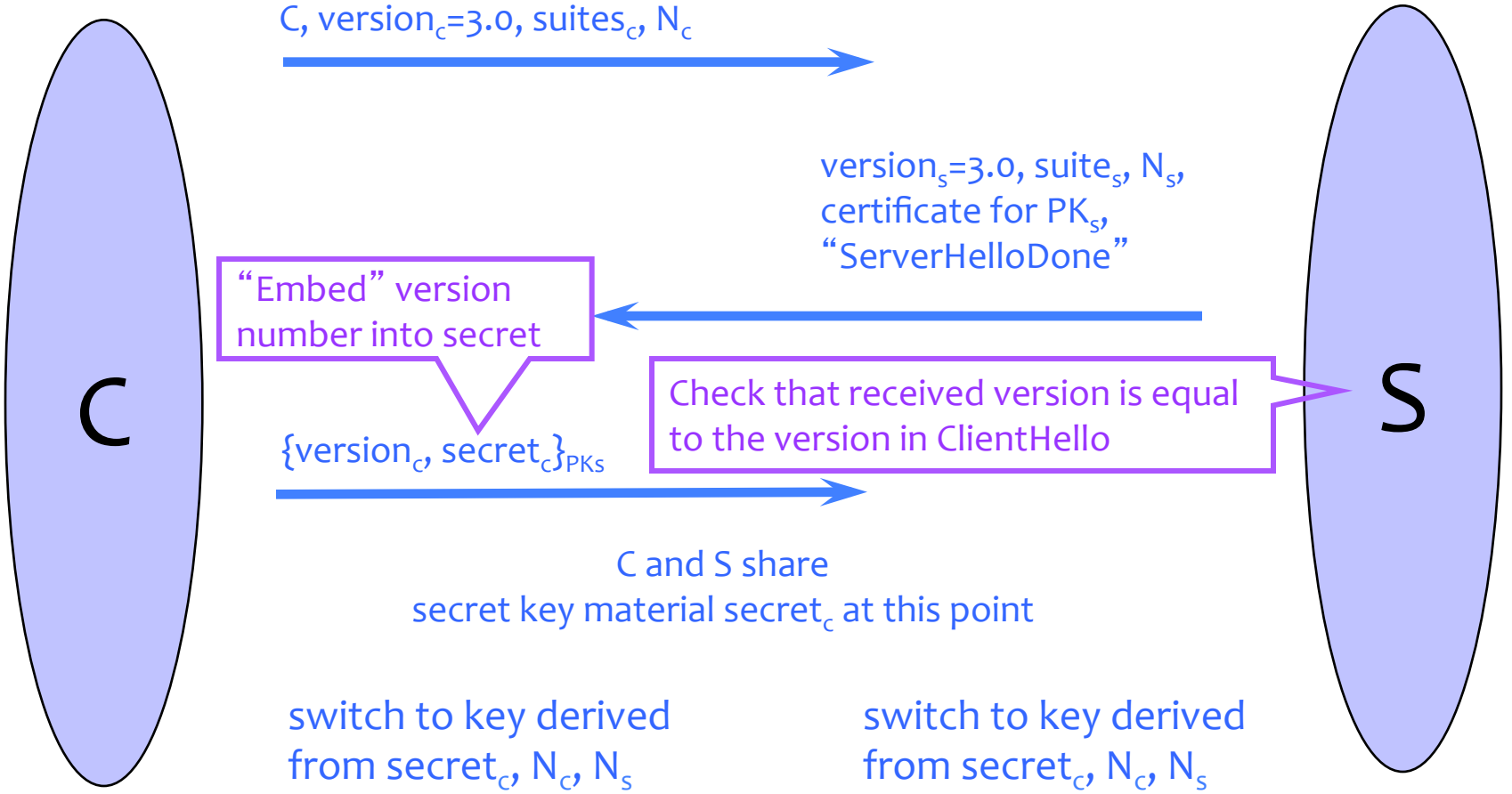


C and S end up communicating using SSL 2.0 (weaker earlier version of the protocol that does not include "Finished" messages)

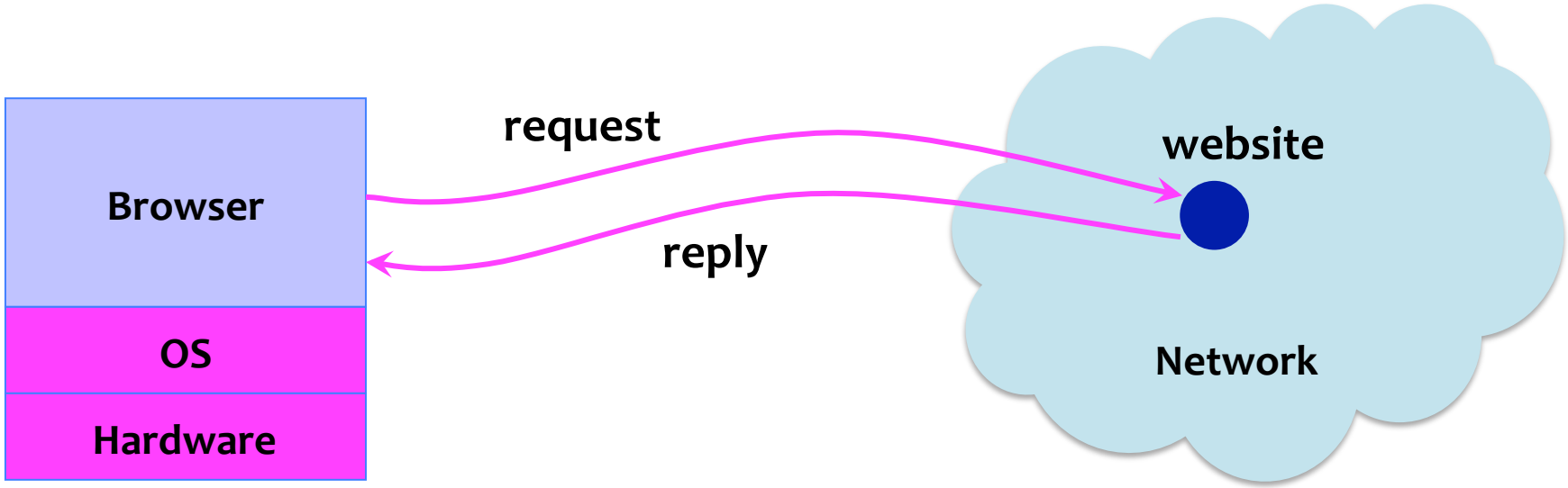
"Chosen-Protocol" Attacks

- Why do people release new versions of security protocols? Because the old version got broken!
- New version must be backward-compatible
  - Not everybody upgrades right away
- Attacker can fool someone into using the old, broken version and exploit known vulnerability
  - Similar: fool victim into using weak crypto algorithms
- Defense is hard: must authenticate version in early designs
- Many protocols have had "version rollback" attacks
  - SSL, SSH, GSM (cell phones)

Version Check in SSL 3.0

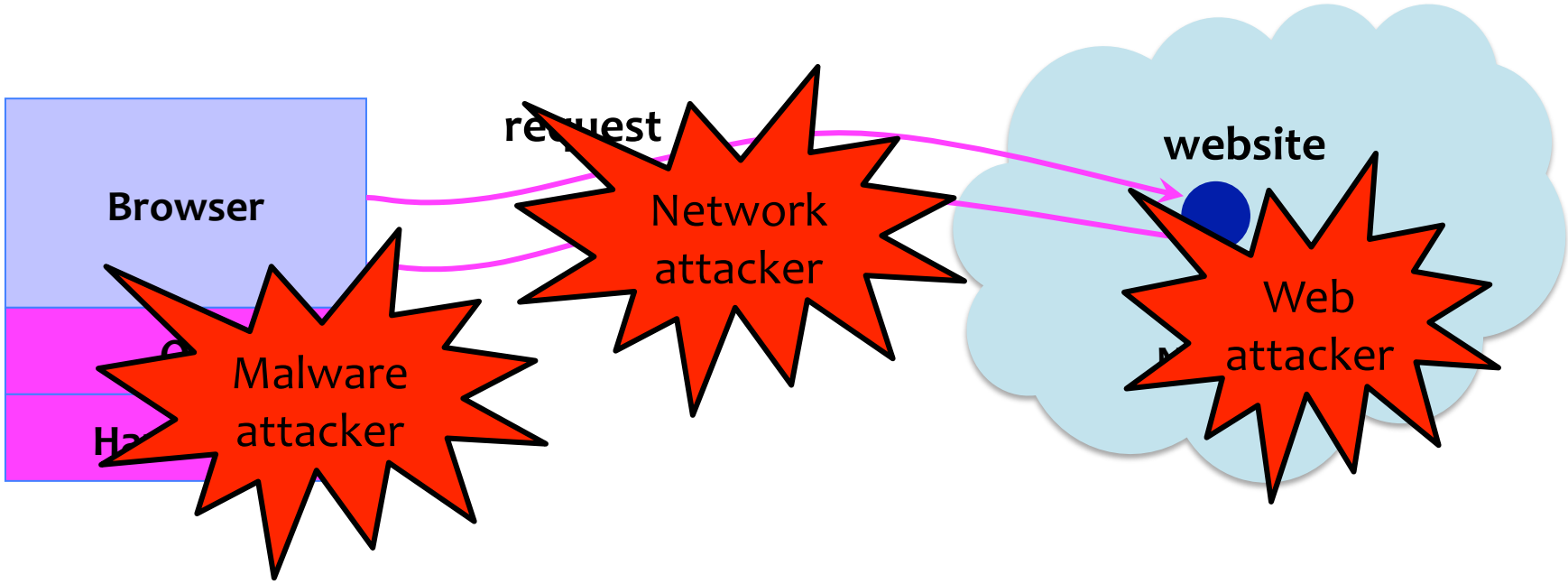


Web Security!
Big Picture: Browser and Network



The browser renders or executes arbitrary HTML, CSS, and Javascript send by hosts on the Internet.

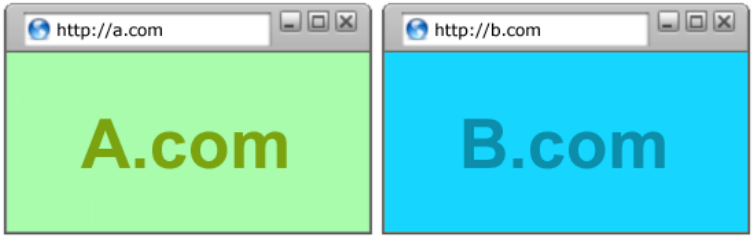
Where Does the Attacker Live?



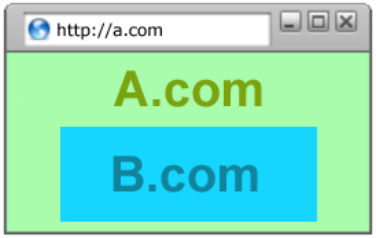
All of These Should Be Safe

- Safe to visit an evil website

- Safe to visit two pages at the same time



- Safe delegation



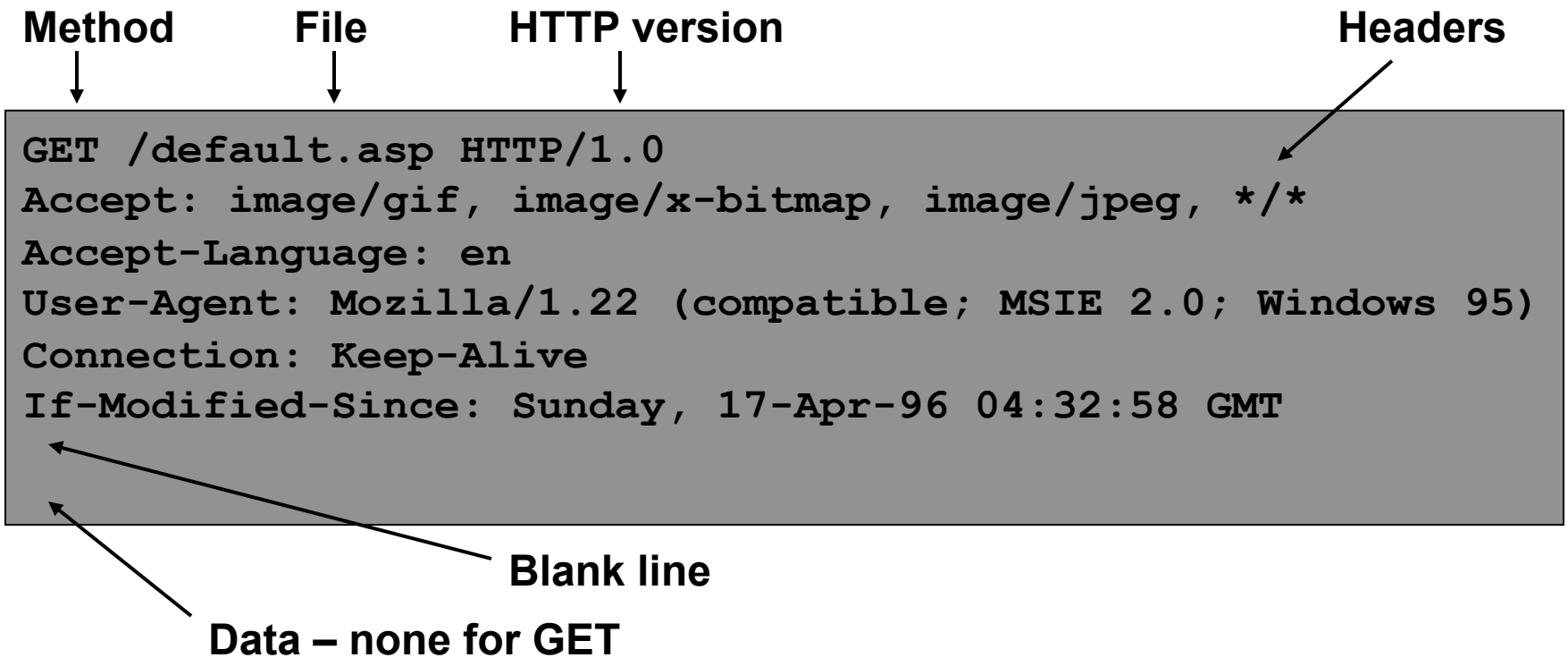
Building Blocks of the Web (and Web Security)

- HTTP(S)
- Cookies

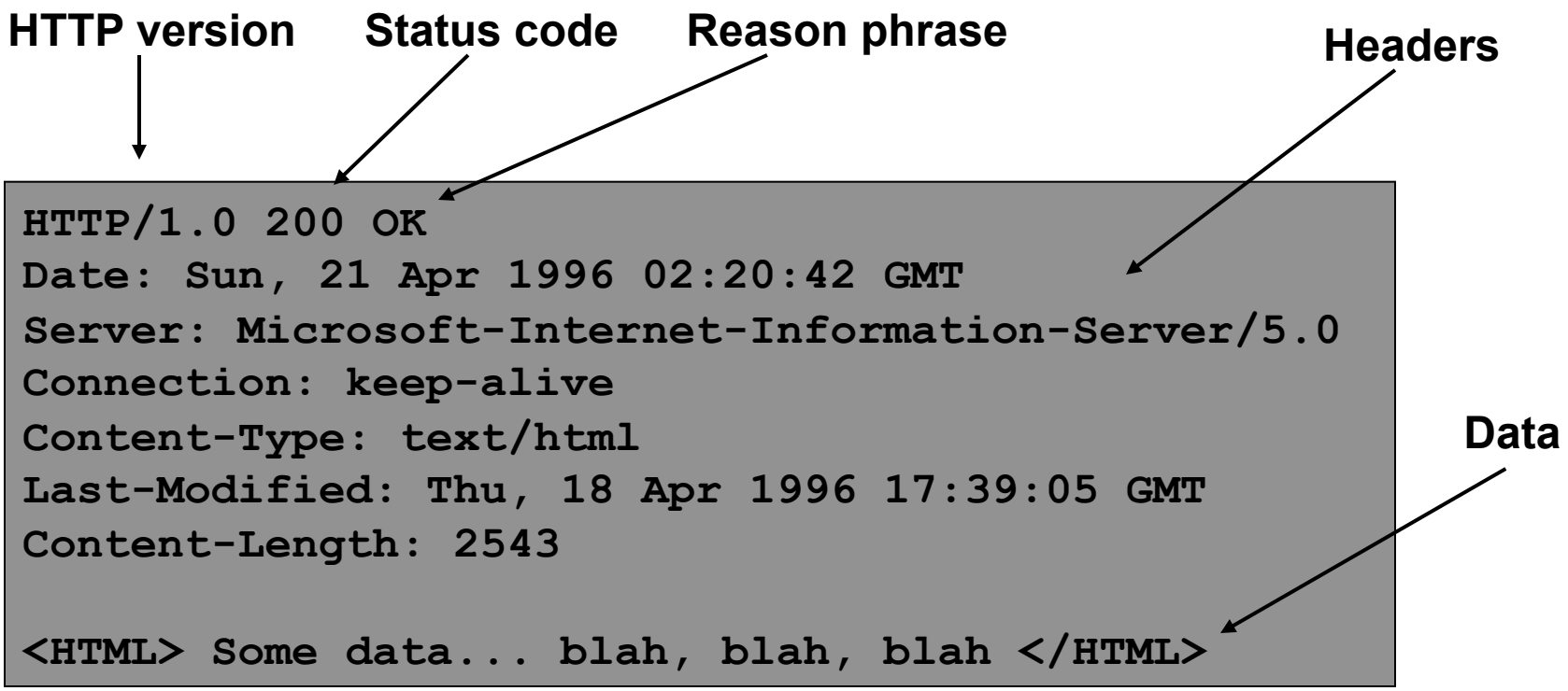
HTTP: HyperText Transfer Protocol

- Application layer protocol used by browsers and web servers
- Stateless request/response protocol
  - Each request is independent of previous requests
  - Statelessness has a significant impact on design and implementation of applications

HTTP Request



HTTP Response



HTTP Verbs

- HTTP declares a number of "verbs" that clients can use to request or provide information
  - **GET** asks for a resource
  - **POST** sends information
  - **HEAD** gets metadata (headers) for a resource
  - Also: PUT, DELETE, TRACE, OPTIONS, CONNECT, PATCH

HTTP Resources

- URL stands for **Uniform Resource Locator**
- Specifies the location of a resource on a network – what server is it on, where is it on that server?

- Resources could include HTML pages, images, data, etc.

HTTP Verbs

- HTTP declares a number of "verbs" that clients can use to request or provide information
  - **GET** asks for a resource
  - **POST** sends information
  - **HEAD** gets metadata (headers) for a resource
  - Also: PUT, DELETE, TRACE, OPTIONS, CONNECT, PATCH

HTTP Verbs

- HTTP declares a number of "verbs" that clients can use to request or provide information
  - **GET** asks for a resource (Give me this image)
  - **POST** sends information
  - **HEAD** gets metadata (headers) for a resource
  - Also: PUT, DELETE, TRACE, OPTIONS, CONNECT, PATCH

HTTP Verbs

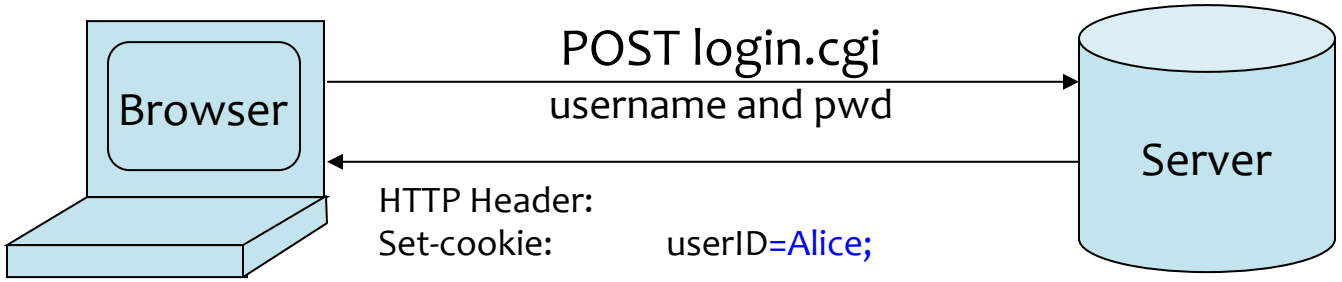
- HTTP declares a number of "verbs" that clients can use to request or provide information
  - **GET** asks for a resource (Give me this image)
  - **POST** sends information (I want to log in)
  - **HEAD** gets metadata (headers) for a resource
  - Also: PUT, DELETE, TRACE, OPTIONS, CONNECT, PATCH

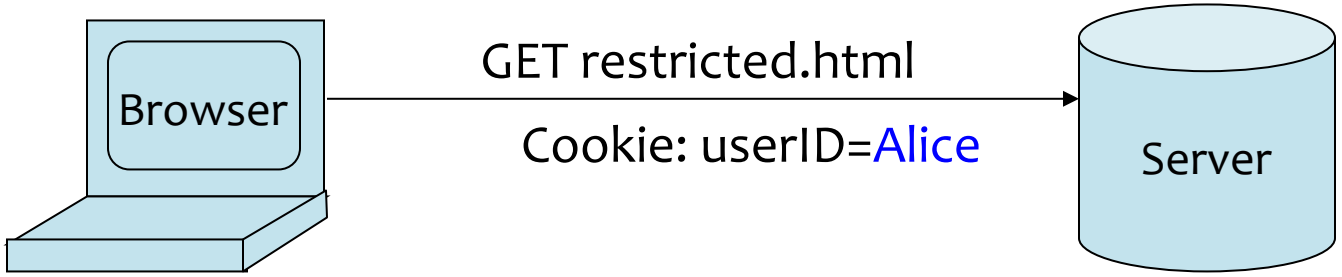
HTTP: HyperText Transfer Protocol

- Application layer protocol used by browsers and web servers
- **Stateless request/response protocol**
  - Each request is independent of previous requests
  - Statelessness has a significant impact on design and implementation of applications

Cookies – Statefulness for HTTP

A cookie is a file created by a website to store information in the browser





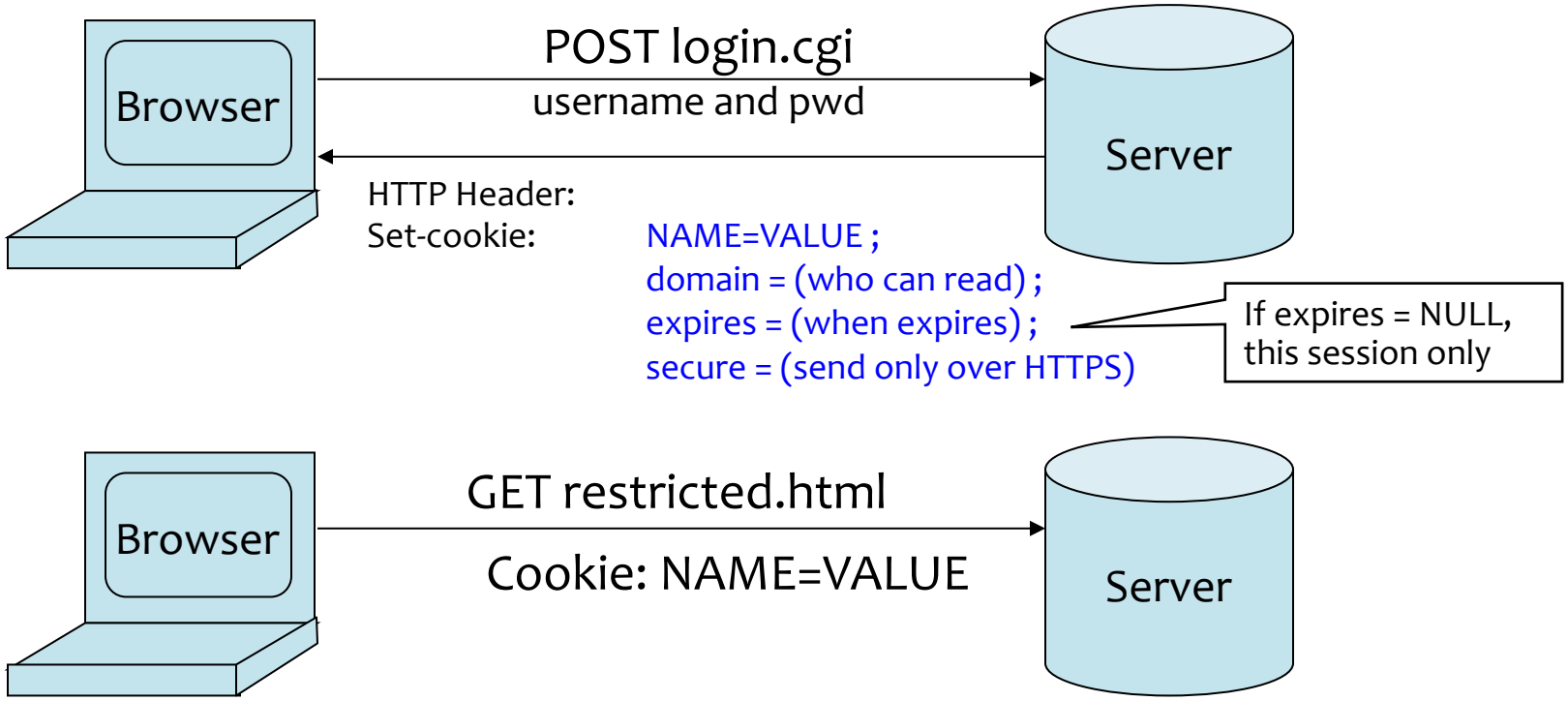
HTTP is a stateless protocol; cookies add state

Cookie Format

- Cookies are just KEY=VALUE pairs, e.g.,
  - `language=ENGLISH`
  - `userID=Alice`
  - `sessionID=8113d906-62e8-49e1-80e1-65805cb51cab`
  - `adID=9c740c60-8d88-4da6-bb83-041e95c1efac`

Cookies – Statefulness for HTTP

A cookie is a file created by a website to store information in the browser



HTTP is a stateless protocol; cookies add state

What Are Cookie Used For?

- Personalization
  - Website remembers visitor preferences
  - `language=ENGLISH`
- Authentication
  - The cookie "proves" client is logged in
  - `sessionID=8113d906-62e8...`
- Tracking
  - Follow the user from site to site;
  - `adID=9c740c60-8d88...`

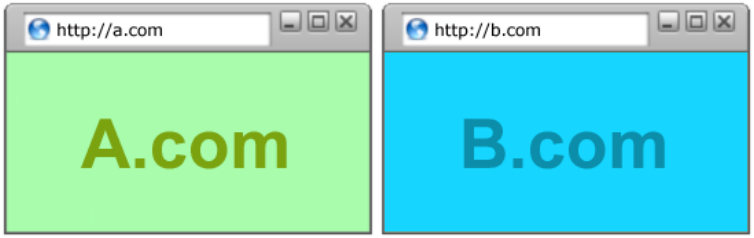
Goals of Web Security

- Safely browse the Web
  - A malicious website cannot steal information from or modify legitimate sites or otherwise harm the user…
  - … even if visited concurrently with a legitimate site -- in a separate browser window, tab, or even iframe on the same webpage
- Support secure Web applications
  - Applications delivered over the Web should have the same security properties we require for standalone applications

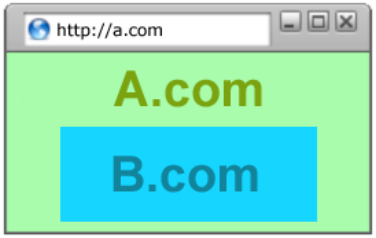
All of These Should Be Safe

- Safe to visit an evil website

- Safe to visit two pages at the same time



- Safe delegation

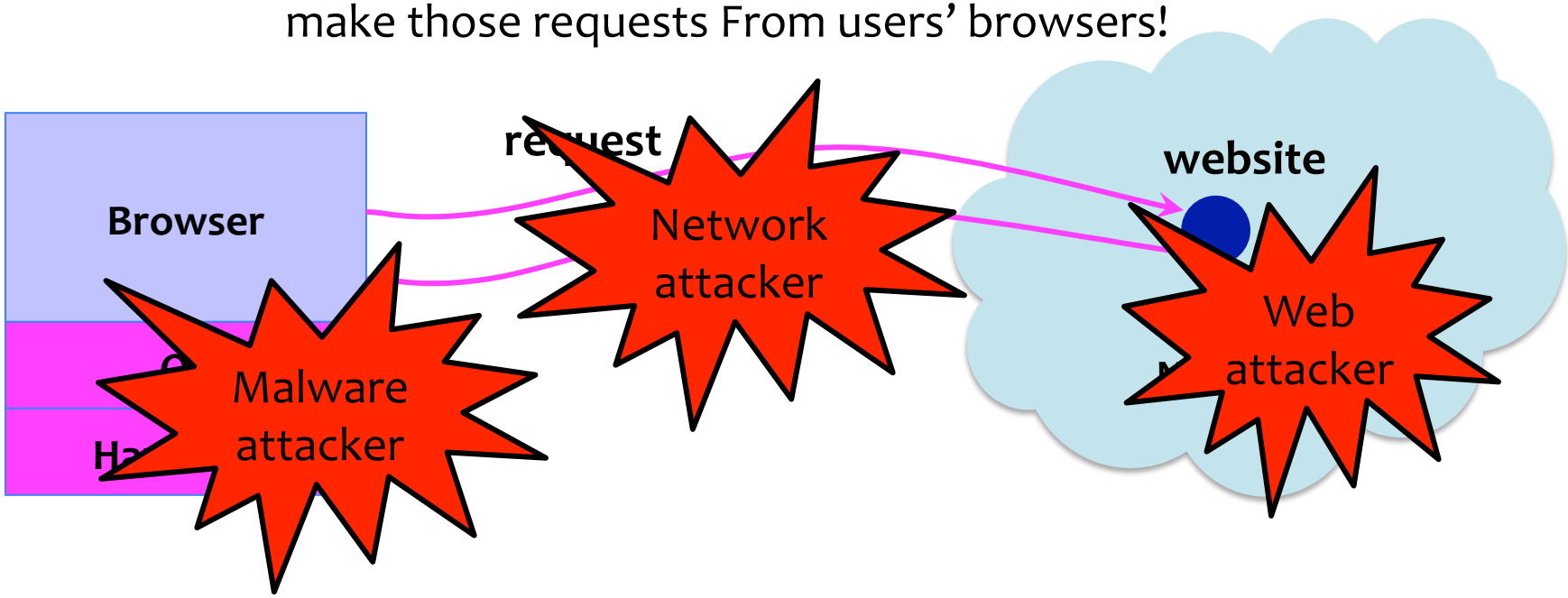


Two Sides of Web Security

- Web browser
  - Responsible for securely confining Web content presented by visited websites
- Web applications
  - Online merchants, banks, blogs, Google Apps …
  - Mix of server-side and client-side code
    - Server-side code written in PHP, Ruby, ASP, JSP… runs on the Web server
    - Client-side code written in JavaScript… runs in the Web browser
  - Many potential bugs: XSS, XSRF, SQL injection

Where Does the Attacker Live?

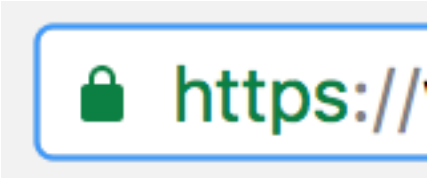
Attacker can make malicious requests to web servers – can even use HTML/JS to make those requests From users' browsers!



Attacker gets to run Javascript and HTML code in the browser

Attacker may control 1 or more domains or websites

Web Attacker



- Controls a malicious website (attacker.com)
  - Can obtain an TLS certificate for attacker.com
- User visits attacker.com – why?
  - Phishing email, enticing content, search results, placed by an ad network, blind luck …
  - Or, attacker.com is embedded on another page – loading the friendly page loads content from attacker.com

Web Attacker

Javascript, or, Software Security for the Web!

Browser receives content, displays HTML and executes scripts

```
<html>
    …
<p> The script on this page is totally trustworthy

<script>

   doSomethingEvil()

</script>
    …
</html>
```

www.attacker.com

A potentially malicious webpage gets to execute some code on user's machine!

Browser Sandbox

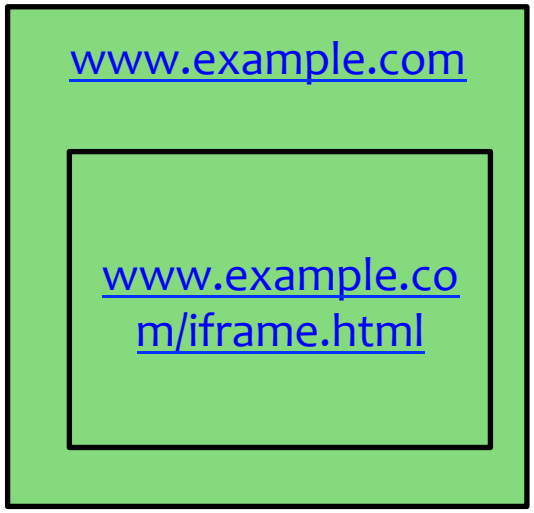
- Goal: safely execute JavaScript provided by a website
  - No/limited access to OS/network/filesystem/browser data.
  - No buffer overflows, no way to execute arbitrary native code, process isolation between tabs
  - Attacker shouldn't be able to access data from other tabs or browser windows
  - attacker.com shouldn't be able to access data from bank.com, even if you're logged in

A Strawperson Attack

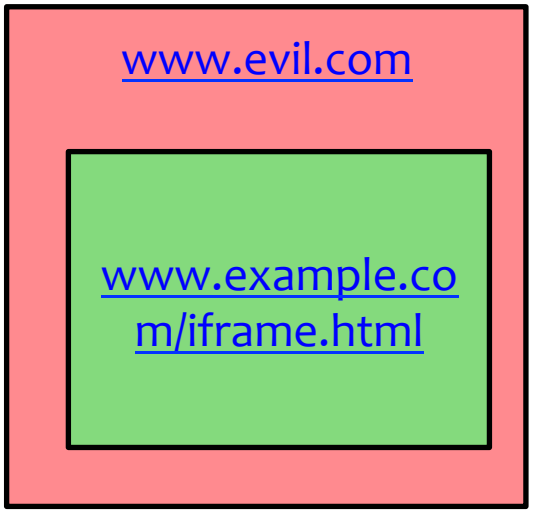
www.attacker.com (the parent) **cannot** access HTML elements in the iframe (and vice versa).

Same-Origin Policy: DOM

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



www.example.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).

Same-Origin Policy

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 thanks to Wikipedia.]