

<https://brown-csci1660.github.io>

# CS1660: Intro to Computer Systems Security Spring 2026

## Lecture 10: Web Security I

Instructor: **Nikos Triandopoulos**

February 26, 2026



BROWN

# CS1660: Announcements

- ◆ Course updates
  - ◆ HW 1 is due today
  - ◆ Lecture notes to be consistently updated
  - ◆ Special materials to be posted for midterm preparation

# Last class



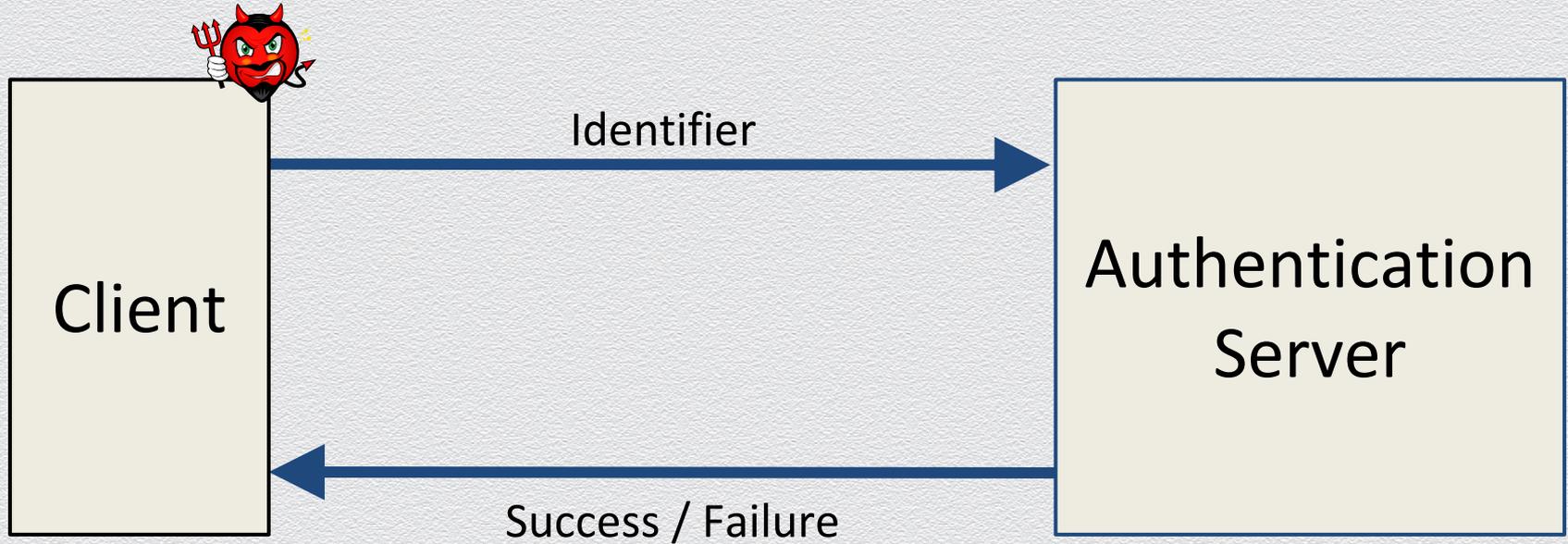
- ◆ Cryptography
  - ◆ Introduction to modern cryptography
  - ◆ Secure communication & symmetric-key encryption in practice
  - ◆ Integrity & reliable communication
  - ◆ Public-key encryption & digital signatures
    - ◆ Motivation, key management, hybrid encryption, implementation, assumptions
- ◆ Authentication
  - ◆ User authentication: something you know, are, have
    - ◆ Password security and cracking, ~~more on password cracking~~
  - ◆ Data authentication: Merkle tree

# Today

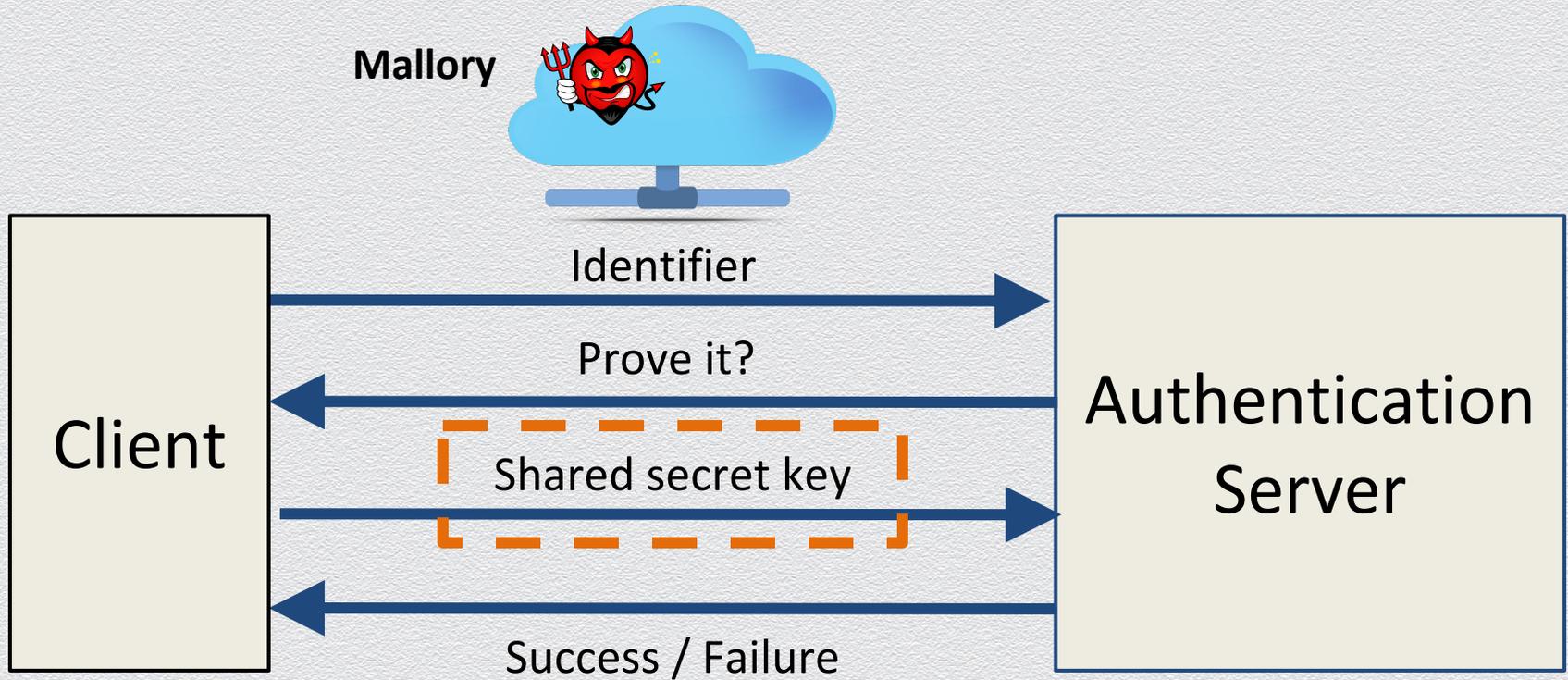
- ◆ Cryptography
  - ◆ Authentication
    - ◆ User authentication: something you know, are, have
    - ◆ Data authentication: Merkle tree
    - ◆ System authentication: Challenge-response methodology, randomness revisited
  - ◆ Web security
    - ◆ The Dyn DDOS attack
    - ◆ Web security model
      - ◆ Background, web-application security, browser security, cookies
-  **Web security**

## **10.0.0 Other authentication protocols**

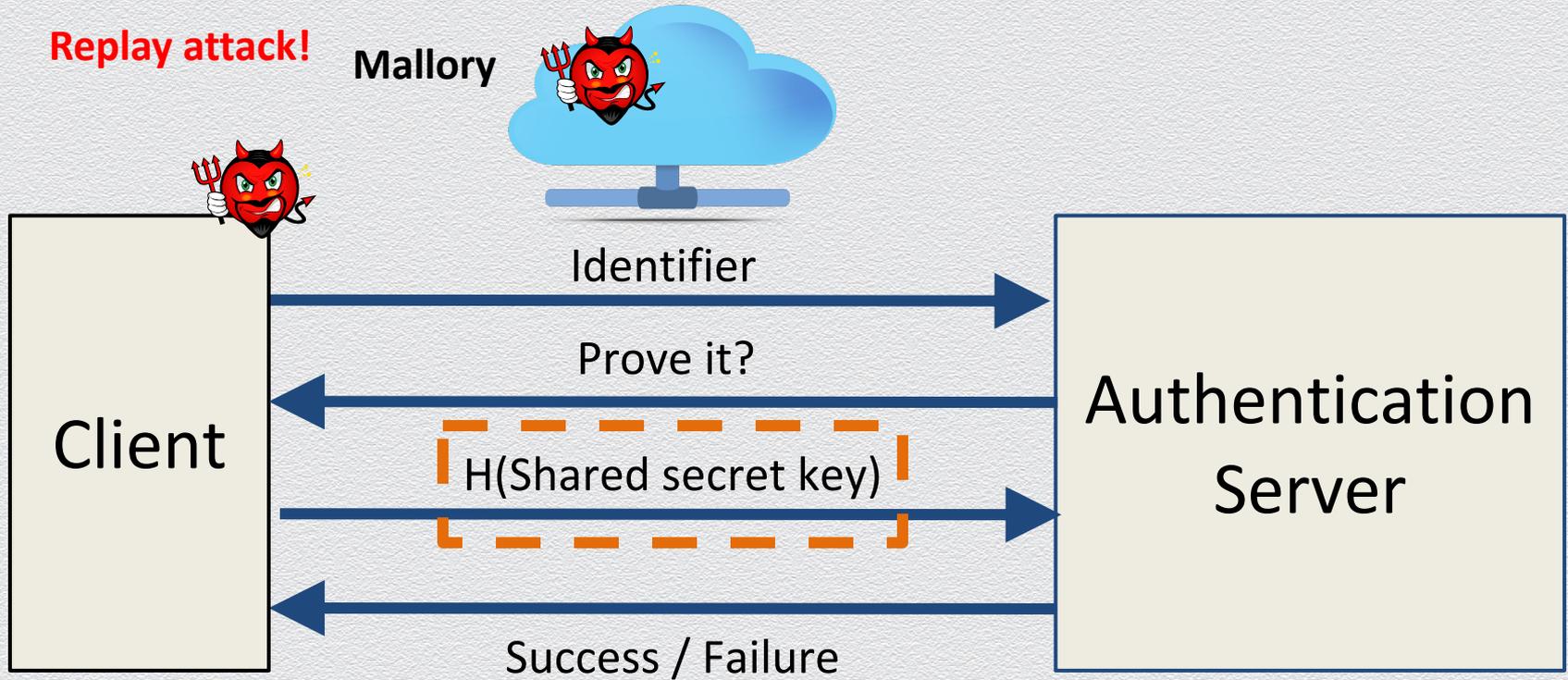
# How to authenticate two systems?



But...



# Even better method...



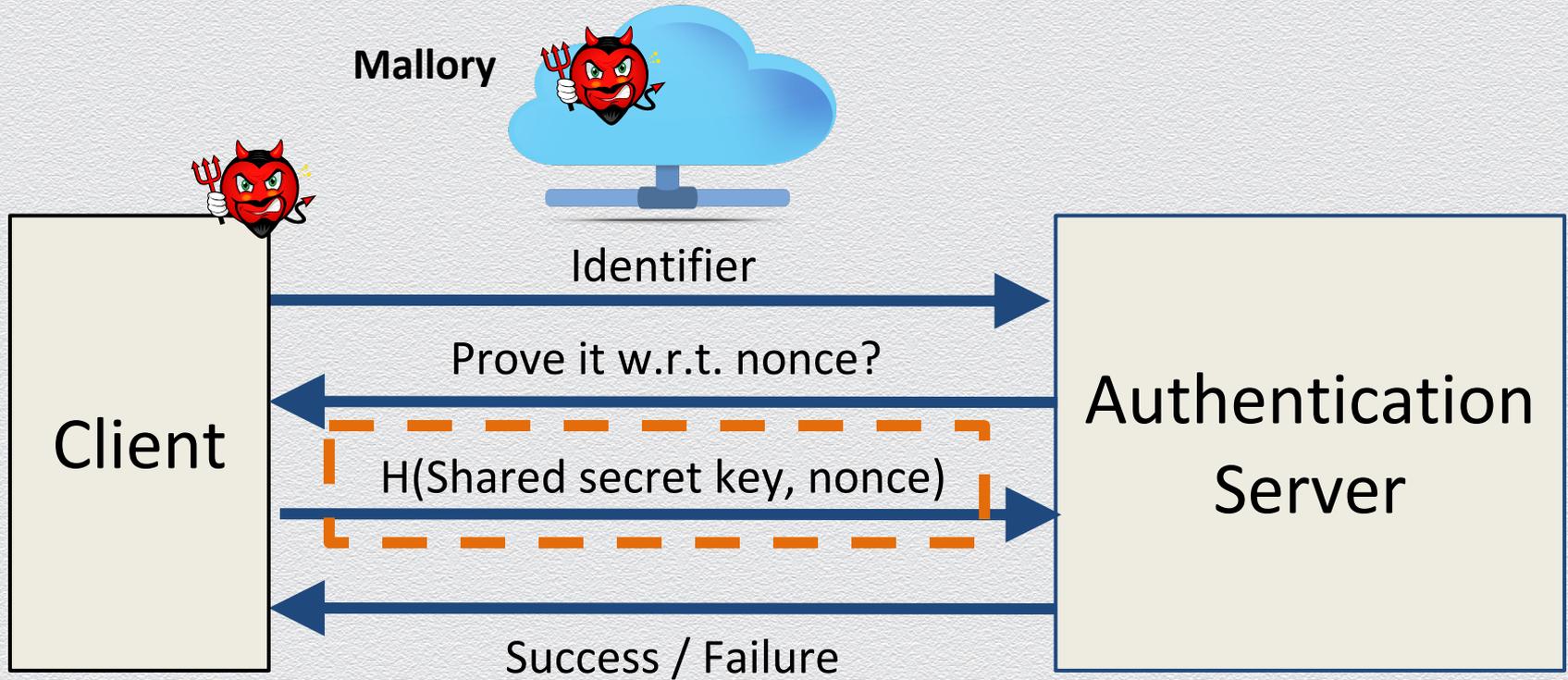
# Challenge-response

- ◆ Use **challenge-response**, to prevent replay attack
  - ◆ Goal is to avoid the reuse of the same credential
- ◆ Suppose Client wants to authenticate Server
  - ◆ **Challenge** sent from Server to Client
- ◆ Challenge is chosen so that...
  - ◆ Replay is not possible
  - ◆ Only Client can provide the correct **Response**
  - ◆ Server can verify the response

# Nonces

- ◆ To ensure “freshness”, can employ a **nonce**
  - ◆ Nonce == **number used once**
- ◆ What to use for nonces?
  - ◆ A **unique** random string
- ◆ What should the Client do with the nonce?
  - ◆ Transform the nonce using the shared secret
- ◆ How can the Server verify the response?
  - ◆ Server knows the shared secret and the nonce, so can check if the response is correct

# Challenge-Response authentication method



# Authentication protocols

- ◆ Challenge response mainly relies on nonce
- ◆ What if nonce wasn't random?
- ◆ Harder to authenticate humans, more on that later...

## **10.0.1 Randomness revisited**

# Entropy

- ◆ Amount of uncertainty in a situation
- ◆ Fair Coin Flip
  - ◆ Maximum uncertainty
- ◆ Biased Coin Flip
  - ◆ More bias → Less uncertainty

# Entropy (cont.)

- ◆ Computers need a source of uncertainty (entropy) to generate random numbers.
  - ◆ Cryptographic keys.
  - ◆ Protocols that need coin flips.
- ◆ Which are sources of entropy in a computer?
  - ◆ Mouse and keyboard movements or thermal noise of processor.
  - ◆ Unix like operating systems use `dev/random` and `dev/urandom` as randomness collector

# Random numbers in practice

- ◆ We need random numbers but...

*“Anyone who considers arithmetical methods of producing random numbers is, of course, in a state of sin.”* - John von Neumann

- ◆ Bootup state is predictable and entropy from the environment may be limited:
  - ◆ Temperature is relatively stable
  - ◆ Oftentimes the mouse/keyboard motions are predictable
- ◆ Routers often use network traffic
  - ◆ Eavesdroppers.
- ◆ Electromagnetic noise from an antenna outside of a building
- ◆ Radioactive decay of a ‘pellet’ of uranium
- ◆ Lava lamps...

# Lava lamps

- ◆ Cloudflare company uses lava lamps as an entropy source



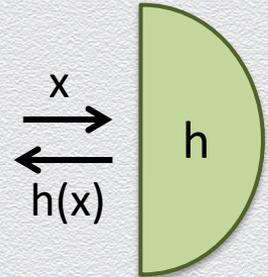
# Provable security: Idealized models

- ◆ challenge in proving security of scheme  $S$  that employs scheme  $S'$ 
  - ◆ no reasonable assumption on  $S'$  or  $\mathcal{A}$  can provide a security proof for  $S$
- ◆ naïve approach: look for other schemes or use scheme  $S$  (if  $S'$  looks “secure”)
- ◆ middle-ground approach: fully rigorous proof Vs. heuristic proofs
  - ◆ employ **idealized** models that **impose** assumptions on  $S'$ ,  $\mathcal{A}$
  - ◆ formally prove security of  $S$  in this idealized model
  - ◆ better than nothing...
- ◆ canonical example: employ the **random-oracle model** when using hashing
  - ◆ a cryptographic hash function  $h$  is treated as a **truly random** function

# The random-oracle model

treats a cryptographic hash function  $h$  as a “black box” realizing a **random** function

- ◆ models  $h$  as a “secret service” that is publicly available for querying
  - ◆ anyone can provide input  $x$  and get output  $h(x)$
  - ◆ nobody knows the exact functionality of the “box”
  - ◆ queries are assumed to be private
- ◆ interpretation of internal processing
  - ◆ if query  $x$  is new, then record and return a **random** value  $h(x)$  in the hash range
  - ◆ otherwise, answer **consistently** with previous queries on  $x$

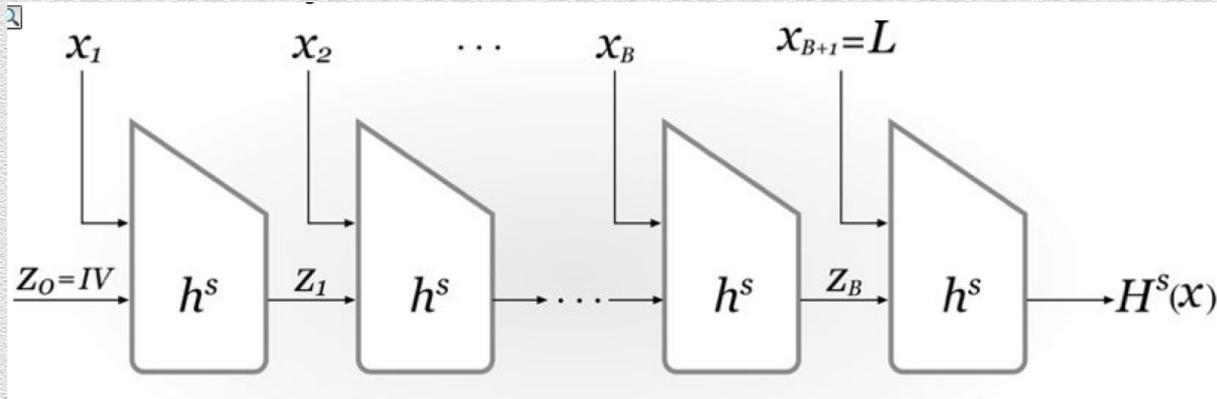


# Random-oracle methodology

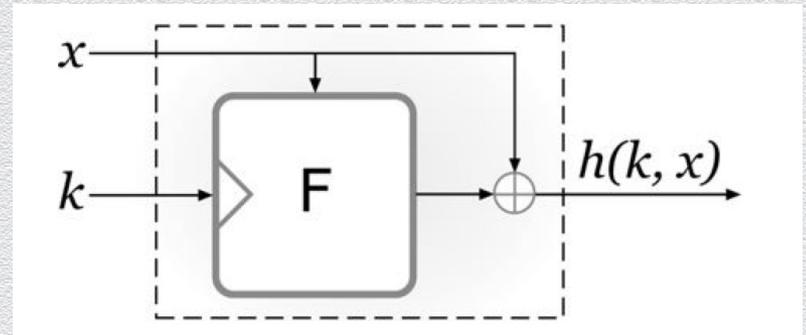
1. design & analyze using random oracle  $h$ ; 2. instantiate  $h$  with specific function  $h'$
- ◆ how sound is such an approach? on-going debate in cryptographic community
  - ◆ pros (proof in random-oracle model better than no proof at all)
    - ◆ leads to significantly **more efficient** (thus practical) schemes
    - ◆ design is **sound**, subject to limitations in instantiating  $h$  to  $h'$
    - ◆ at present, only **contrived** attacks against schemes proved in this model are known
  - ◆ cons (proofs in the standard model are preferable)
    - ◆ random oracles **may not exist** (cannot deterministically realize random functions)
    - ◆ real-life  $\mathcal{A}$ s see the code of  $h'$  (e.g., may find a shortcut for some hash values)
    - ◆ can construct scheme  $S$ , s.t.  $S$  is proven secure using  $h$ , but is insecure using  $h'$
    - ◆ note: “ $h'$  is CR” Vs. “ $h'$  is a random oracle”

# Recall: Constructing hash functions in practice

## Merkle-Damgård transform



## The Davies-Meyer scheme

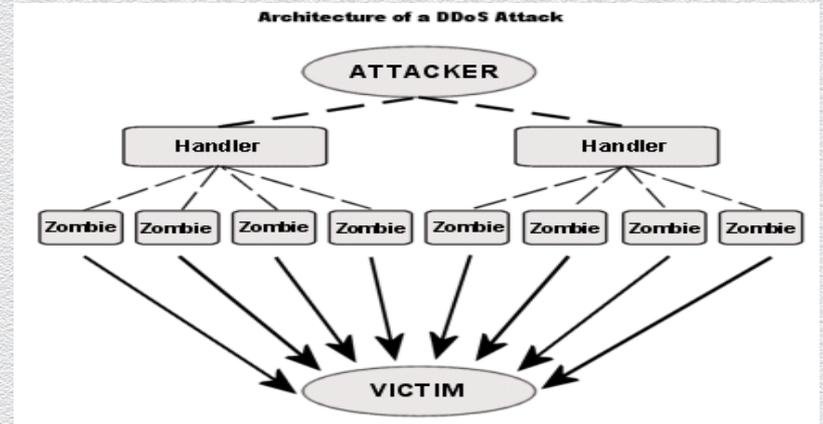
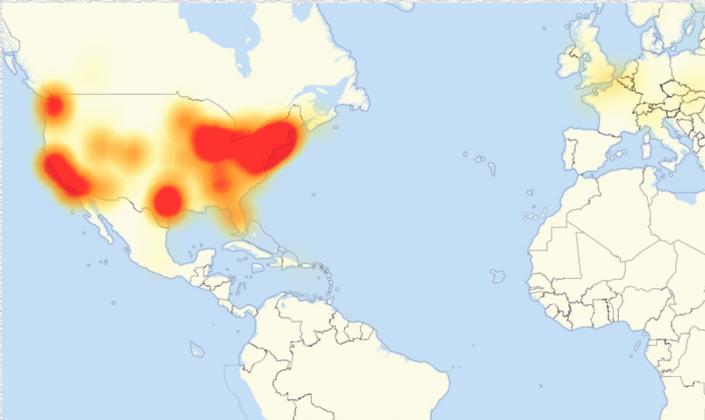


## **10.1 The Dyn DDoS attack**

# It's unfair! – I had no class but couldn't watch my Netflix series!

On October 21, 2016, a large-scale cyber was launched

- ◆ it affected globally the entire Internet but particularly hit U.S. east coast
- ◆ during most of the day, no one could access a long list of major Internet platforms and services, e.g., Netflix, CNN, Airbnb, PayPal, Zillow, ...
- ◆ this was a **Distributed Denial-of-Service (DDoS)** attack



# Domain Name Service (DNS) protocol

Resolving domain names to IP addresses

- ◆ when you type a URL in your Web browser, its IP address must be found
  - ◆ larger websites have multiple IP responses for redundancy to distributing load
- ◆ at the heart of Internet addressing is a protocol called DNS
  - ◆ a database translating Internet names to addresses



query: Please resolve netflix.com

←

→

answer: IP is 52.22.118.132



# DNS: Hierarchical search

Search is performed recursively and hierarchically across different type of DNS resolvers

- ◆ Untrusted recursive DNS servers: query other resolvers and cache recent results
- ◆ Trusted TLD (top-level domain) servers: control TLD zones such as .com, .org, .net, etc.

**DNS entries:**

<netflix.com, 52.22.118.132>



primary

**subset of cached queried entries**

(or information of other resolvers)



secondary

25

**locally cached IP addresses**

(at Web browser and OS)

netflix.com

52.22.118.132  
(or “non-existent”)

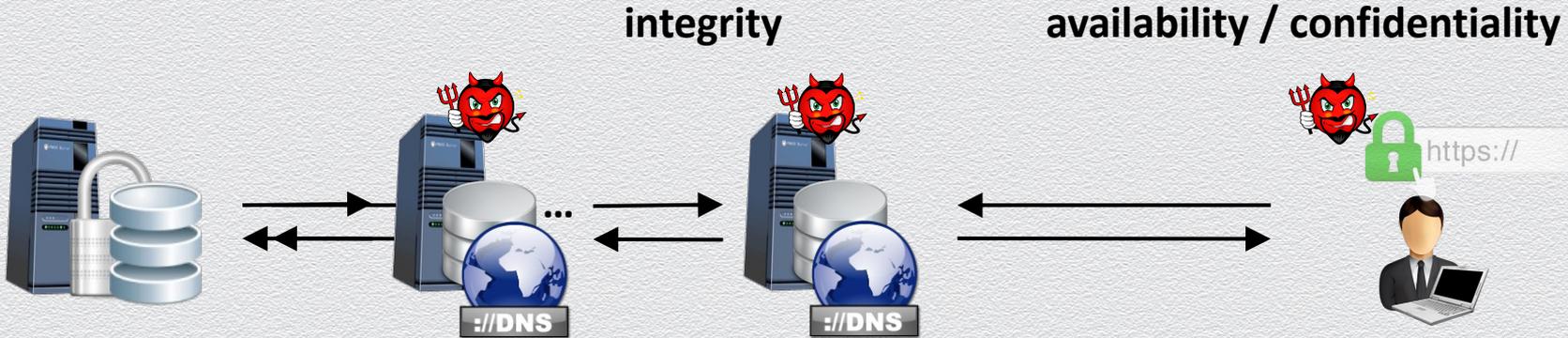
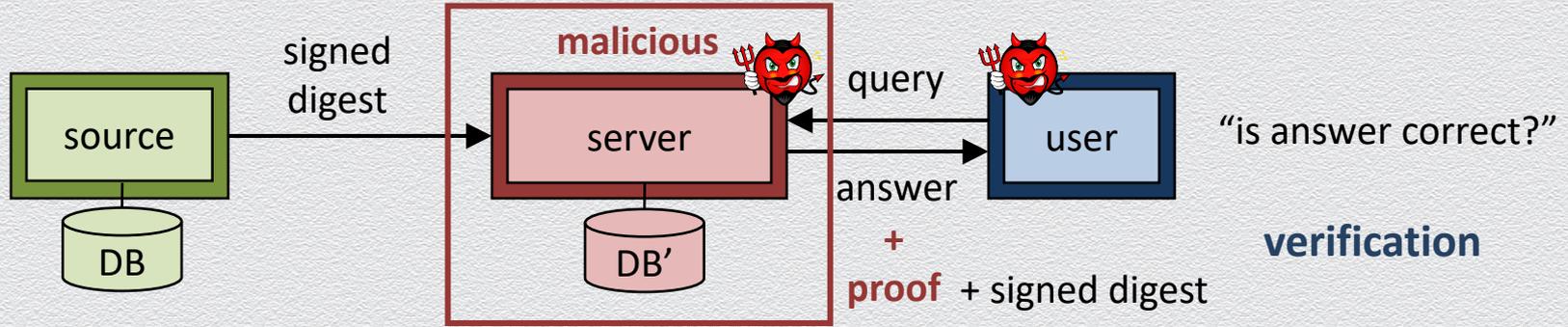


# DNS: A critical asset to attack...

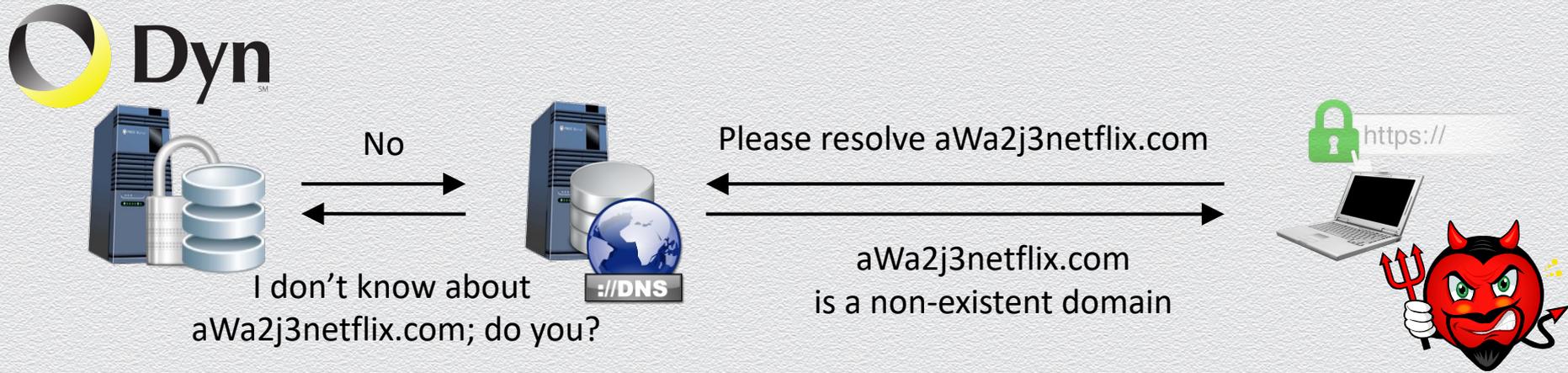
What main security properties must be preserved in such an important service?

- ◆ all properties in CIA triad are relevant!
- ◆ resolving domain names to IP addresses is a service that
  - ◆ must critically be available during all times – availability
  - ◆ must critically be trustworthy – integrity
  - ◆ must also protect database entries that are not queried – confidentiality

# DNS: A critical asset to attack... (cont.)



# Dyn DDoS attack



## Attack:

- ◆ from a compromised machine ask for domain names that do not exist
- ◆ query is forwarded to fewer primary Dyn servers, i.e., defeating benefits of distribution
- ◆ use a botnet to ask **A LOT** of such queries to bring down the Dyn DNS service!

# Dyn DDoS attack: Exploit Internet of Things (IoT)



No



I don't know about  
aWa2j3netflix.com; do you?



Please resolve aWa2j3netflix.com



aWa2j3netflix.com  
is a non-existent domain



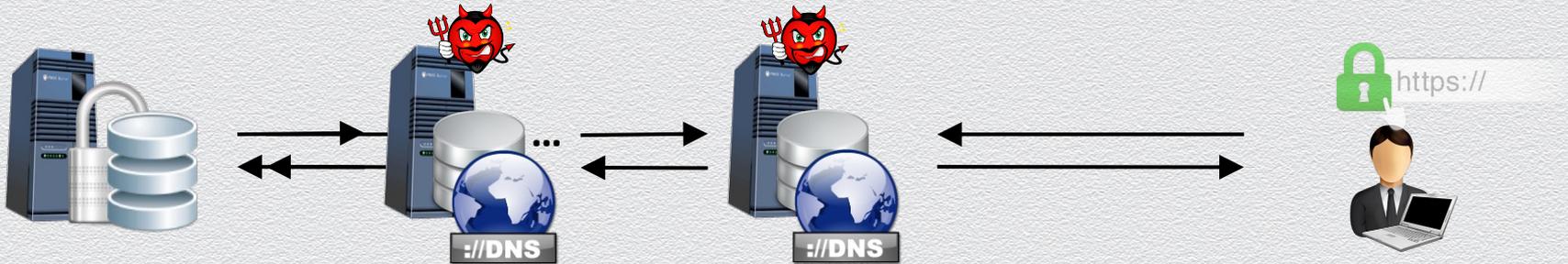
## Create a botnet:

- ◆ compromise easy targets: IoT “thin” devices, e.g., printers, cameras, home routers, ...
- ◆ how? find a vulnerability on these devices...
- ◆ all such devices used an OS with a static, hard-wired, thus known, admin password...!

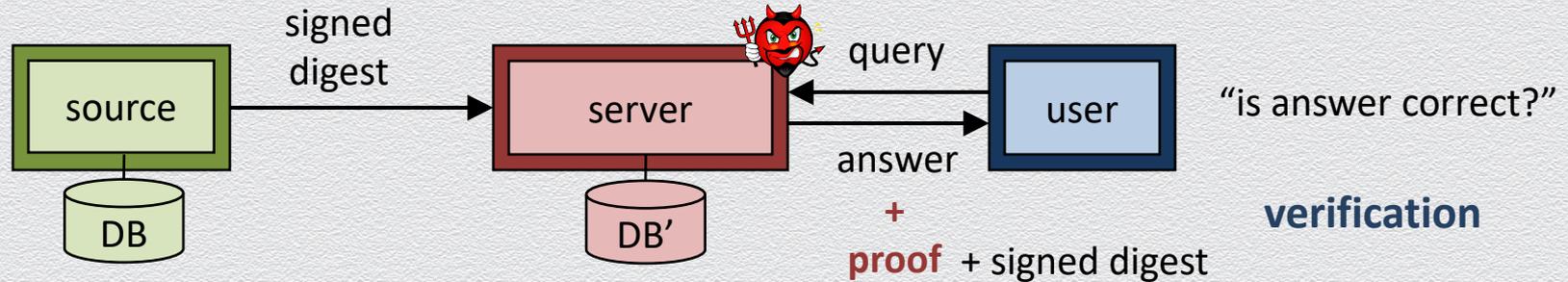
# DNSSEC & NSEC

Security extensions of DNS protocol to protect integrity of DNS data

- ◆ correct resolution, origin authentication, authenticated denial of existence
- ◆ specifications made by Internet Engineering Task Force (IETF) via RFCs
  - ◆ an RFC (request for comments) is a suggested solution under peer review
- ◆ challenges: backward-compatible, simplicity, confidentiality, who signs
  - ◆ DNSSEC/NSEC: extension that provide proofs of existence/denial of existence



# DNSSEC & NSEC: core idea



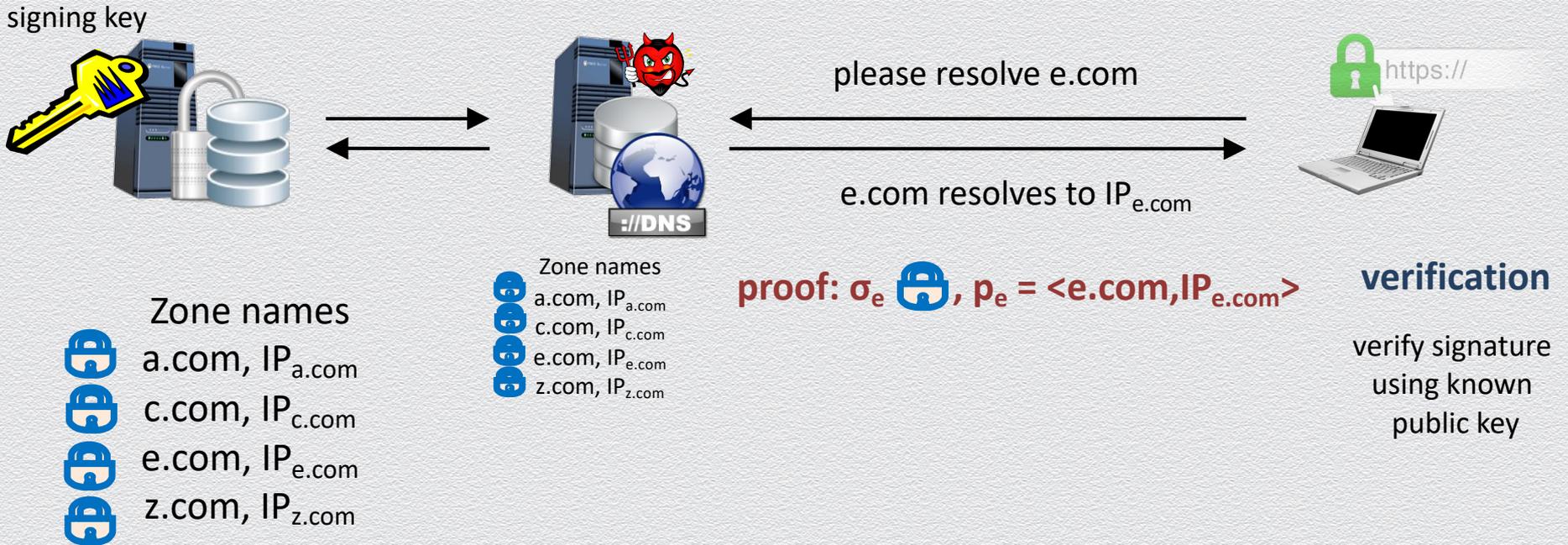
**DNSSEC protocol:** each DNS entry is pre-signed by primary name server

**NSEC protocol:**

- domain names are lexicographically ordered and then each pair of neighboring existing domain names is pre-signed by the primary name server
- non-existing names, e.g., aWa2j3netflix.com are proved by providing this pair "containing" missed query name, e.g., <awa.com, awb.com>

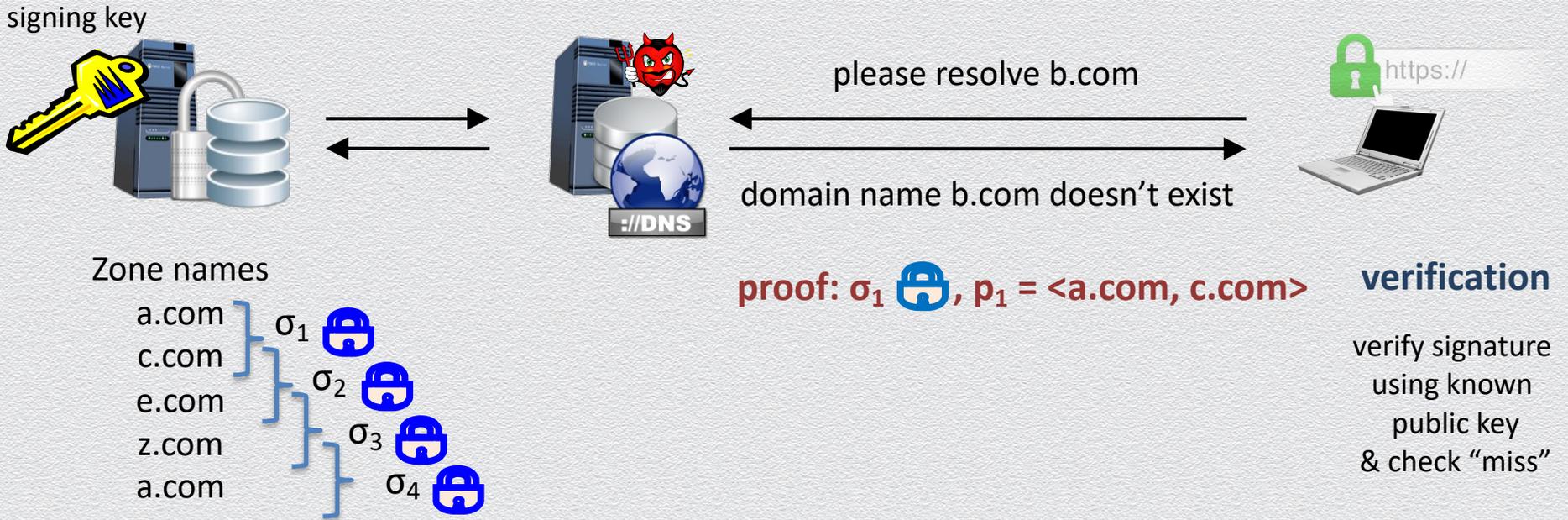
# DNSSEC: example

Each entry <domain name, IP address> in the database is individually signed by a primary DNS server and uploaded to secondary DNS servers in signed form



# NSEC: example

Additionally, pairs of consecutive (in alphabetical order) domain names are individually signed by a primary DNS server and uploaded to secondary DNS servers in signed form



# NSEC: Vulnerability

exploit the “leak-domain-names”  
vulnerability of NSEC to learn the  
domain names of an entire zone

Proofs of non-existing names leak information about other unknown domain names

signing key



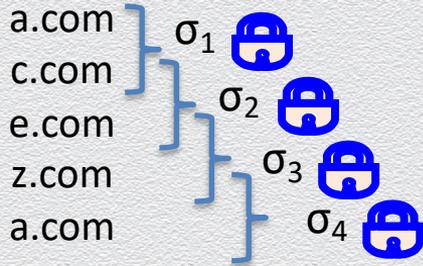
please resolve b.com



domain name b.com doesn't exist



Zone names



proof:  $\sigma_1$  ,  $p_1 = \underline{\langle a.com, c.com \rangle}$

leaked information

user asked for b.com but  
also learned for a.com & c.com

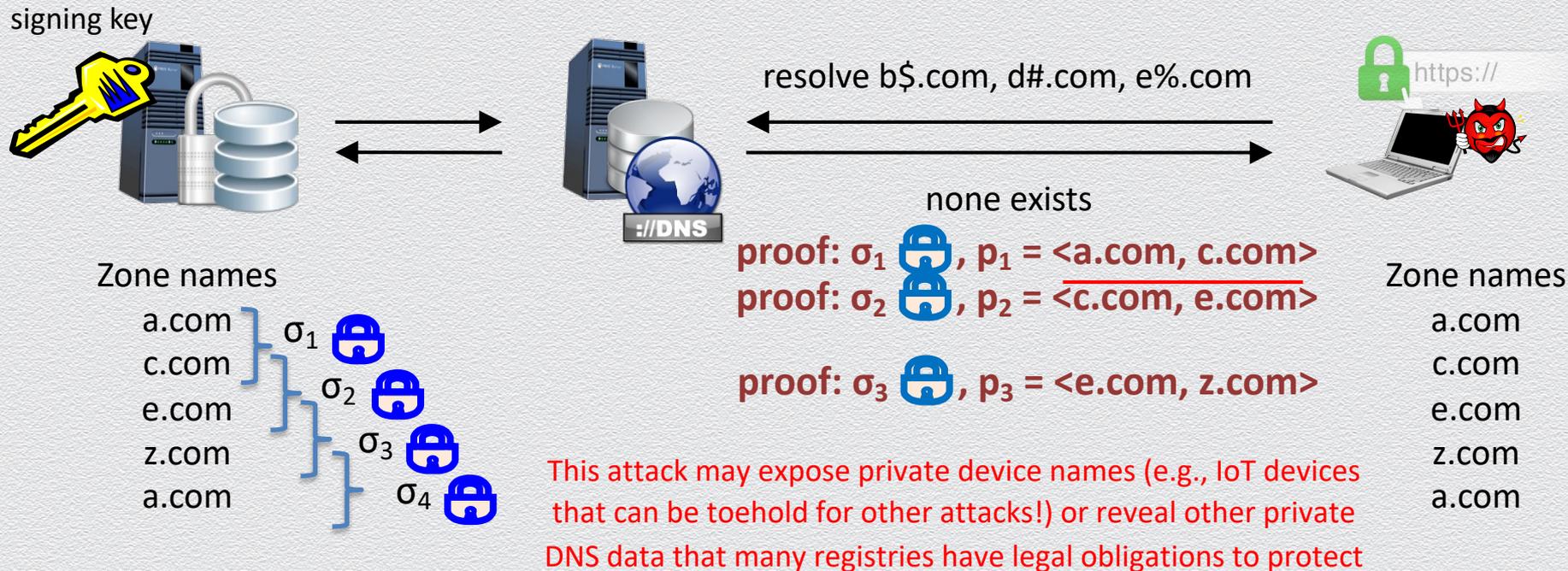
verification

verify signature  
using known  
public key  
& check “miss”

# Zone enumeration attack

ask for non-existing names  
to get all possible proofs

An attacker can simply act as a “querier” to learn target organization’s network structure!



# NSEC3: NSEC in the hash domain



Zone names



asked for b.com but  
learned h(e.com) & h(z.com)

$h(b.com) = \text{ntwo4}$   
e.g., h is SHA-256

# NSEC5: A secure solution



Zone names



$h'(x)$ , RSA-signature of  $f(b.com)$

asked for b.com but learned  $h'(e.com)$  &  $h'(z.com)$

$h'(b.com) = ntwo4$

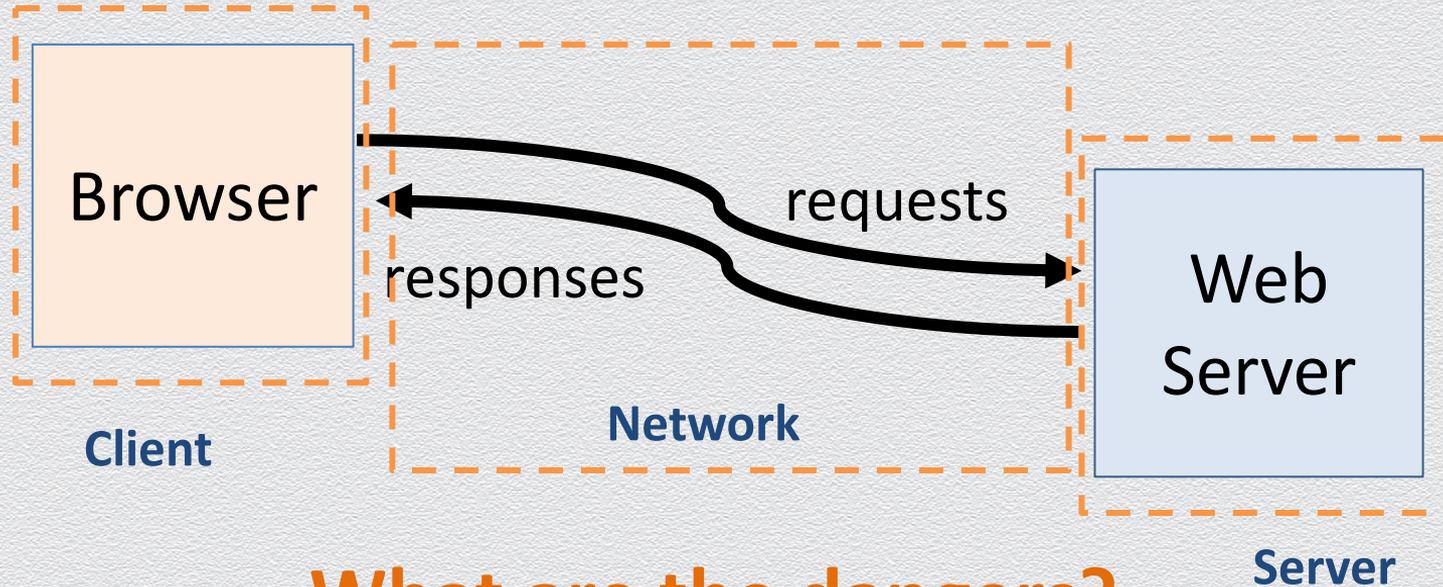
h: as in NSEC3

f: "message transformation" hash

$$h'(x) = h( \text{RSA-Sign}( \text{key icon}, f(x) ) )$$

## **10.2 Web security model**

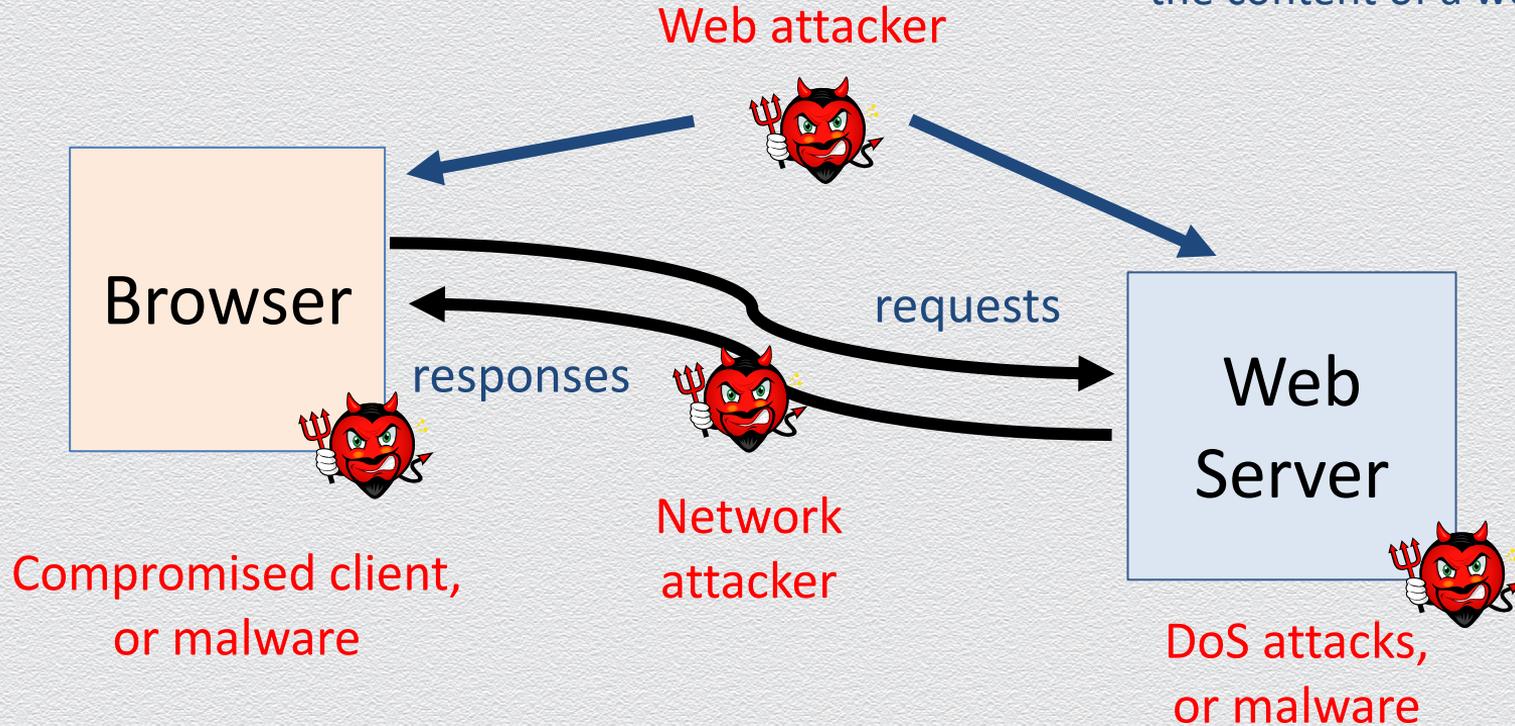
# Web applications



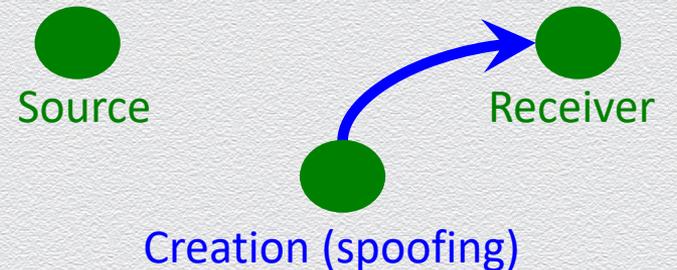
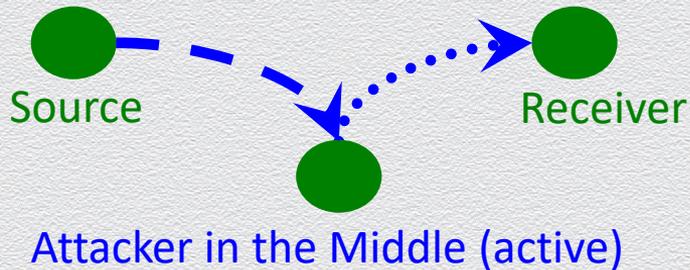
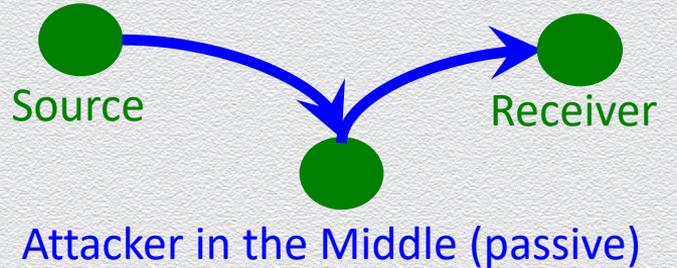
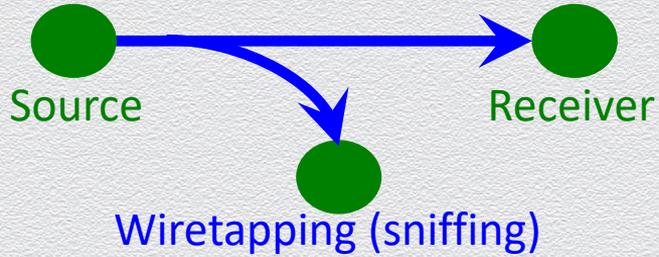
**What are the dangers?**

# Threat models

The main vector of attack is via the content of a website



# Network attacks



# Web Attacker Capabilities

- ◆ Attacker controls a malicious website
  - ◆ website might look professional, legitimate, etc.
  - ◆ attacker can get users to visit website (how?)
- ◆ A benign website is compromised by attacker
  - ◆ attacker inserts malicious content into website
  - ◆ attacker steals sensitive data from website
- ◆ Attacker does not have direct access to user's machine

# Potential Damage

- ◆ An attacker gets you to visit a malicious website...
  - ◆ Can they perform actions on other websites impersonating you?
  - ◆ Can they run evil code on your OS?
- ◆ Ideally, none of these exploits are possible ...

# Attack Vectors

- ◆ Web browser (focus of this lecture)
  - ◆ Renders web content (HTML pages, scripts)
  - ◆ Responsible for confining web content
  - ◆ **Note:** Browser implementations dictate what websites can do
- ◆ Web applications
  - ◆ Server code (PHP, Ruby, Python, ...)
  - ◆ Client-side code (JavaScript)
  - ◆ Many potential bugs (e.g., see Project 2)

# Browser Security: Sandbox

Goal: protect local computer from web attacker

- ◆ Safely execute code on a website, without the code
  - ◆ accessing your files, tampering with your network, or accessing other sites

High stakes

- ◆ \$40K bounty for Google Chrome
  - ◆ [www.google.com/about/appsecurity/chrome-rewards/](http://www.google.com/about/appsecurity/chrome-rewards/)

We won't address attacks that break the sandbox

- ◆ But they happen check the CVE list
  - ◆ <https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=sandbox>
  - ◆ <https://support.apple.com/en-us/HT213635>

## **10.3 Domains, HTML, HTTP**

# URL and FQDN

## URL: Uniform Resource Locator

`https://cs.brown.edu/about/contacts.html`

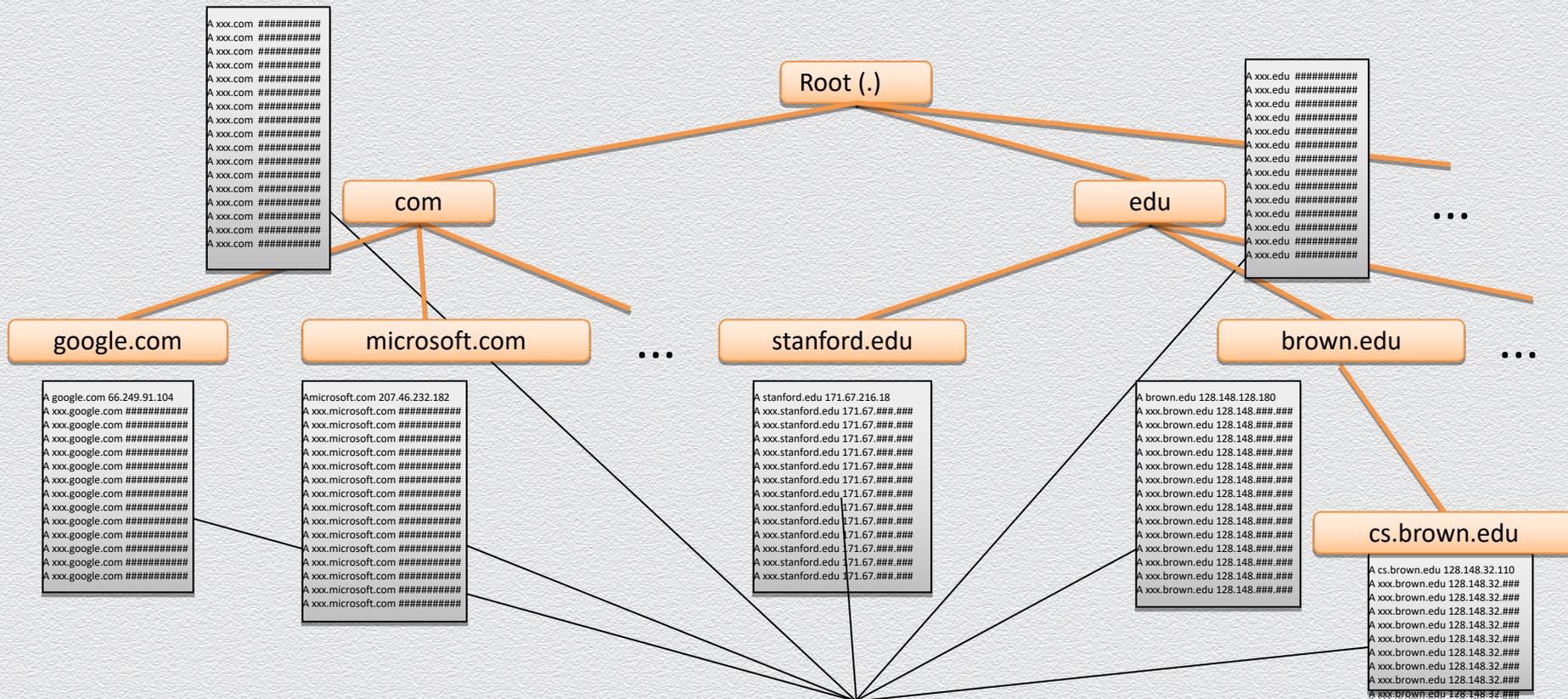
- ◆ a protocol
  - ◆ e.g. https
- ◆ a FQDN
  - ◆ e.g. cs.brown.edu
- ◆ a path and file name
  - ◆ e.g. /about/contacts.html

## FQDN: Fully Qualified Domain Name

[Host name].[Domain].[TLD].[Root]

- ◆ Two or more labels, separated by dots
  - ◆ e.g., cs.brown.edu
- ◆ Root name server
  - ◆ a “.” at the end of the FQDN
- ◆ Top-level domain (TLD)
  - ◆ generic (gTLD): .com, .org, .net,
  - ◆ country-code (ccTLD): .ca, .it, , .gr ...

# Domain hierarchy



# HTML

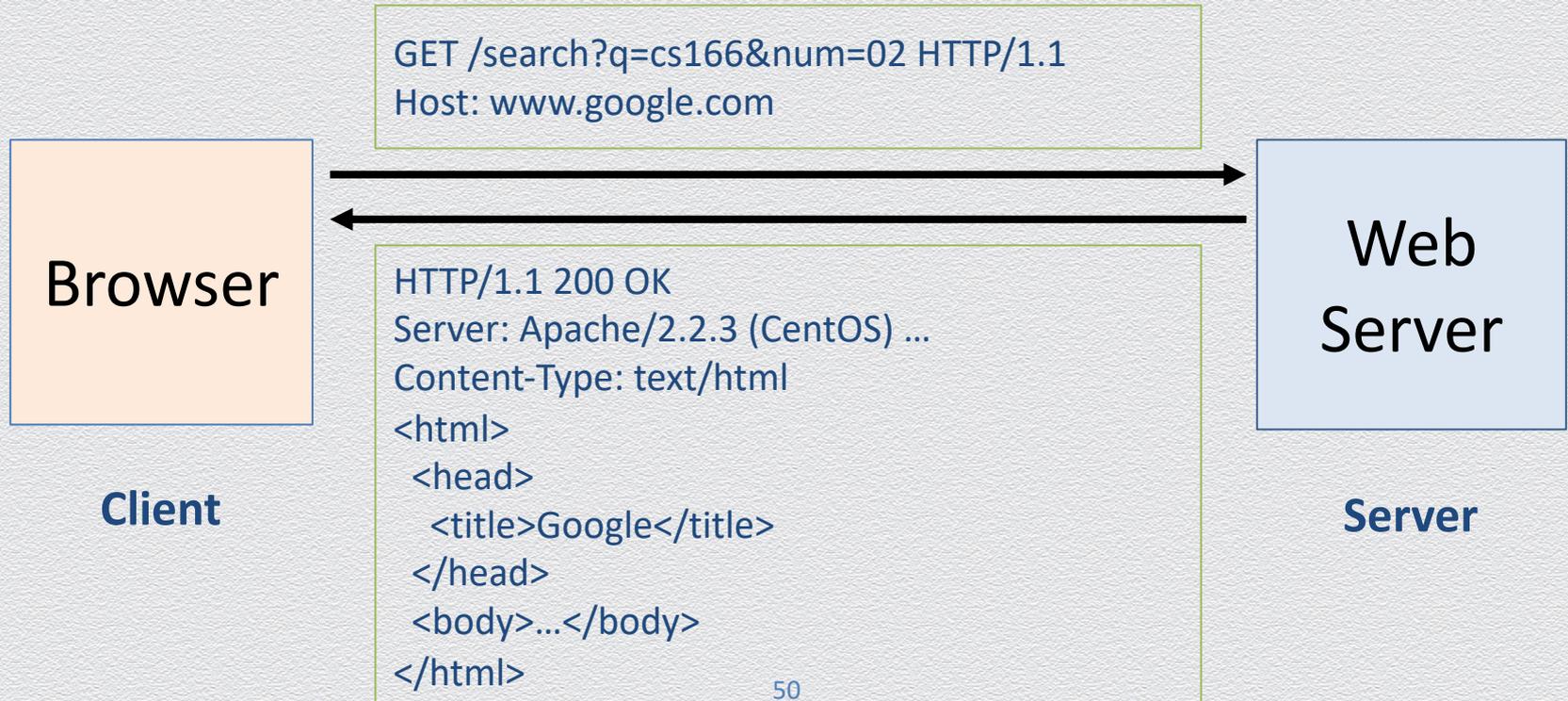
## Hypertext markup language (HTML)

- ◆ allows linking to other pages (href)
- ◆ supports embedding of images, scripts, other pages (script, iframe)
- ◆ user input accepted in forms

```
<html>
  <head>
    <title>Google</title>
  </head>
  <body>
    <p>Welcome to my page.</p>
    <script>alert("Hello world");
    </script>
    <iframe src="http://example.com">
    </iframe>
  </body>
</html>
```

# HTTP (Hypertext Transport Protocol)

Communication protocol between client and server



# What's in a request (or response)?

URL (domain, path)

Variables (name-value pairs)

```
GET /search?q=cs166&num=02 HTTP/1.1
Host: www.google.com
```

Browser

Web Server

```
HTTP/1.1 200 OK
Server: Apache/2.2.3 (CentOS) ...
Content-Type: text/html
<html>
  <head>
    <title>Google</title>
  </head>
  <body>...</body>
</html>
```

Resource

# Variables

Key-value pairs obtained from user input into forms & submitted to server

- ◆ Submit variables in HTTP via GET or POST
- ◆ GET request: variables within HTTP URL
  - ◆ e.g., `http://www.google.com/search?q=cs166&num=02`
- ◆ POST request: variables within HTTP body
  - ◆ POST / HTTP/1.1
  - ◆ Host: `example.com`
  - ◆ Content-Type: `application/x-www-form-urlencoded`
  - ◆ Content-Length: `18`
  - ◆ `month=5&year=2024`

# Semantics: GET Vs. POST

## GET

- ◆ Request target resource
- ◆ Read-only method
- ◆ Submitted variables may specify target resource and/or its format

## POST

- ◆ Request processing of target resource
- ◆ Read/write/create method
- ◆ Submitted variables may specify how resource is processed
  - ◆ e.g., content of resource to be created, updated, or executed

# GET Vs. POST

	<b>GET</b>	<b>POST</b>
Browser history	✓	X
Browser bookmarking	✓	X
Browser caching	✓	X
Server logs	✓	X
Reloading page	immediate	warning
Variable values	Restricted	arbitrary

## **10.4 Web-application security**

# Client-side controls

- ◆ Web security problems arises because clients can submit arbitrary input
- ◆ What about using client-side controls to check the input?
- ◆ Which kind of controls?

# Client-side controls (cont.)

A standard application may rely on client-side controls

- ◆ They restrict user input in two general ways
  - ◆ Transmitting data via the client component using a mechanism that should prevent the user from modifying that data
  - ◆ Implementing measures on the client side
- ◆ In this threat model
  - ◆ Server does not trust the Client

# Bypassing client-side controls

- ◆ In general, a security flaw because it is easy to bypass
- ◆ The user
  - ◆ has a full control over the client and the data it submits
  - ◆ can bypass any controls that are client-side and not replicated on the server
- ◆ Why these controls are still useful?
  - ◆ For load balancing or usability
  - ◆ Often we can suppose that the vast majority of users are honest

# Transmitting data via the client

- ◆ A common developer bad habit is passing data to the client in a form that the end user cannot directly see or modify
- ◆ Why is it so common?
  - ◆ It removes or reduces the amount of data to store server side per-session
  - ◆ In multi-server applications, it removes the need to synchronize the session data among different servers
  - ◆ The use of third-party components on the server may be difficult or impossible to integrate
- ◆ Transmitting data via the client is often the easy solution
  - ◆ But unfortunately it is not secure

# Common mechanisms

- ◆ HTML Hidden fields
  - ◆ A field flagged hidden is not displayed on-screen
- ◆ HTTP Cookies
  - ◆ Not displayed on-screen, and the user cannot modify directly
- ◆ Referrer Header
  - ◆ An optional field in the http request that it indicates the URL of the page from which the current request originated
- ◆ If you use the proper tool you can tamper the data on the client-side

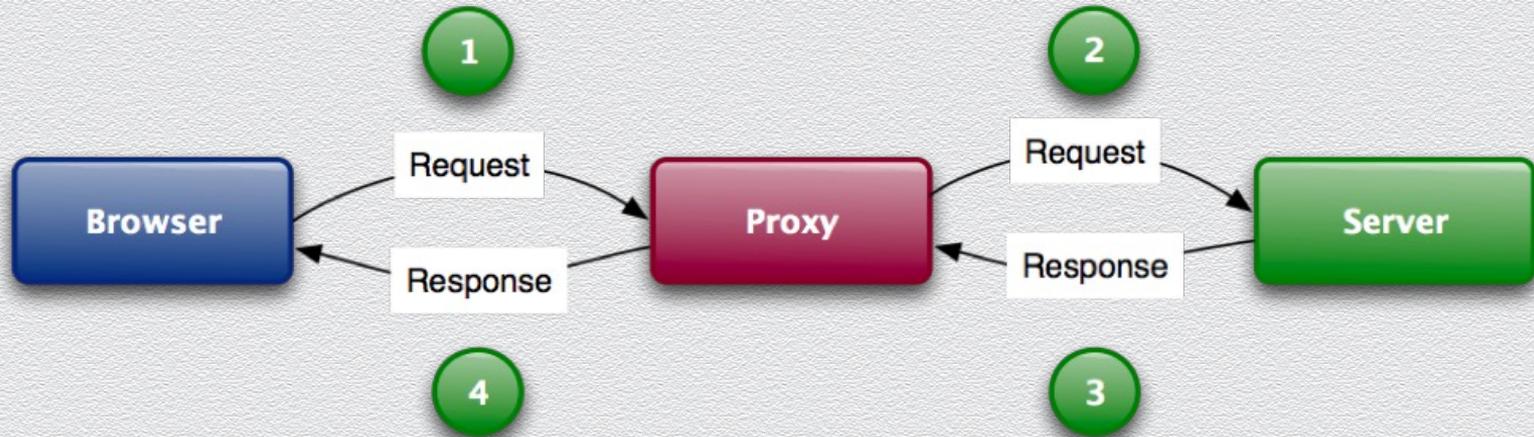
# Web client tool

- ◆ Web inspection tool:
  - ◆ Firefox or Chrome web developer:
    - ◆ powerful tools that allow you to edit HTML, CSS and view the coding behind any website: CSS, HTML, DOM and JavaScript
- ◆ Web Proxy:
  - ◆ Burp, OWASP ZAP, etc.
    - ◆ Allow to modify GET or POST requests

# HTTP proxy

An intercepting Proxy:

- ◆ inspect and modify traffic between your browser and the target application
  - ◆ Burp Intruder, OWASP ZAP, etc.



## **10.5 Browser security**

# In BROWSER we trust...

- ◆ Most of our trust on web security relies on information stored in the Browser
  - ◆ a Browser should be updated since Bugs in the browser implementation can lead to various attacks
  - ◆ e.g., <https://us-cert.cisa.gov/ncas/current-activity/2023/02/14/mozilla-releases-security-updates-firefox-110-and-firefox-esr>
- ◆ Add-ons too are dangerous
  - ◆ Hacking Team flash exploits - [goo.gl/syVwiD](https://goo.gl/syVwiD)
  - ◆ [github.com/greatsuspender/thegreatsuspender/issues/1263](https://github.com/greatsuspender/thegreatsuspender/issues/1263)
- ◆ Executing a browser with low privileges helps

# Browser Security: Same-Origin Policy (SOP)

Very simple idea: “Content from different origins should be isolated”

- ◆ Website origin defined over tuple (protocol, domain, port)

Very difficult to execute in practice...

- ◆ Messy number of cases to worry about...

HTML elements, Navigating Links, Browser cookies, JavaScript capabilities, iframes, ...  
etc.

- ◆ Browsers didn't always get this correct...

# Browser Security: Same-Origin Policy (SPO) (cont.)

Goal: Protect and isolate web content from other web content

- ◆ Content from different origins should be isolated, e.g., mal.com should not interact with bank.com in unexpected ways
- ◆ What about cs.brown.edu vs brown.edu or mail.google.com vs drive.google.com?
- ◆ Lots of subtleties

# SOP example: `http://store.company.com/dir/page.html`

(protocol, domain, port)

URL	Outcome	Reason
<code>http://store.company.com/dir2/other.html</code>	Same origin	Only the path differs
<code>http://store.company.com/dir/inner/another.html</code>	Same origin	Only the path differs
<code>https://store.company.com/page.html</code>	Failure	Different protocol
<code>http://store.company.com:81/dir/page.html</code>	Failure	Different port ( <code>http://</code> is port 80 by default)
<code>http://news.company.com/dir/page.html</code>	Failure	Different host

## 10.6 Cookies

# Cookies

- ◆ HTTP is a stateless protocol; cookies used to emulate state
- ◆ Servers can store cookies (name-value pairs) into browser
  - ◆ user preferences (e.g., language and page layout), user tracking, authentication
  - ◆ expiration date can be set
  - ◆ may contain sensitive information (e.g., for user authentication)

```
POST /login.php HTTP/1.1  
Set-Cookie: Name: sessionid  
            Value: 19daj3kdop8gx  
            Domain: cs.brown.edu  
            Expires: Wed, 21 Feb 2024 ...
```

- ◆ Browser sends back cookies to server on the next connection

# Cookies scope

Each cookie has a scope

- ◆ base domain, which is a given host
  - ◆ e.g., brown.edu
- ◆ plus, optionally, all its subdomains
  - ◆ cs.brown.edu, math.brown.edu, www.cs.brown.edu , etc.
  - ◆ for ease of notation, included subdomains are denoted as .
    - ◆ e.g., .brown.edu
    - ◆ in fact, specified in HTTP with the "Domain:" attribute of a cookie

# Same Origin Policy: Cookie Reads

Websites can only read cookies within their scope

- ◆ Browser has cookies with scope
  - ◆ brown.edu
  - ◆ .brown.edu
  - ◆ .math.brown.edu
  - ◆ cs.brown.edu
  - ◆ .cs.brown.edu
  - ◆ blog.cs.brown.edu
- ◆ Browser accesses cs.brown.edu
- ◆ Browser sends cookies with scope
  - ◆ .brown.edu
  - ◆ cs.brown.edu
  - ◆ .cs.brown.edu

# Same Origin Policy: Cookie Writes

A website can set cookies for (1) its base domain;  
or (2) a super domain (except TLDs) and its subdomains

- ◆ Browser accesses cs.brown.edu
- ◆ cs.brown.edu can set cookies for
  - ◆ .brown.edu
  - ◆ cs.brown.edu
- ◆ But not for
  - ◆ google.com
  - ◆ .com
  - ◆ .math.brown.edu
  - ◆ brown.edu
  - ◆ ...

# Session Management

## Session

- ◆ keep track of client over a series of requests
- ◆ server assigns clients a unique, unguessable ID
- ◆ clients send back ID to verify themselves

## Session

- ◆ necessary in sites with authentication
  - ◆ e.g., banking
- ◆ useful in most other sites
  - ◆ e.g., remembering preferences
- ◆ various methods to implement them
  - ◆ mainly cookies
  - ◆ but also could be in HTTP variables

# Session Management (cont.)

- ◆ Goal
  - ◆ users should not have to authenticate for every single request
- ◆ Problem
  - ◆ HTTP is stateless
- ◆ Solution
  - ◆ user logs in once
  - ◆ server generates session ID and gives it to browser
    - ◆ temporary token that identifies and authenticates user
  - ◆ browser returns session ID to server in subsequent requests

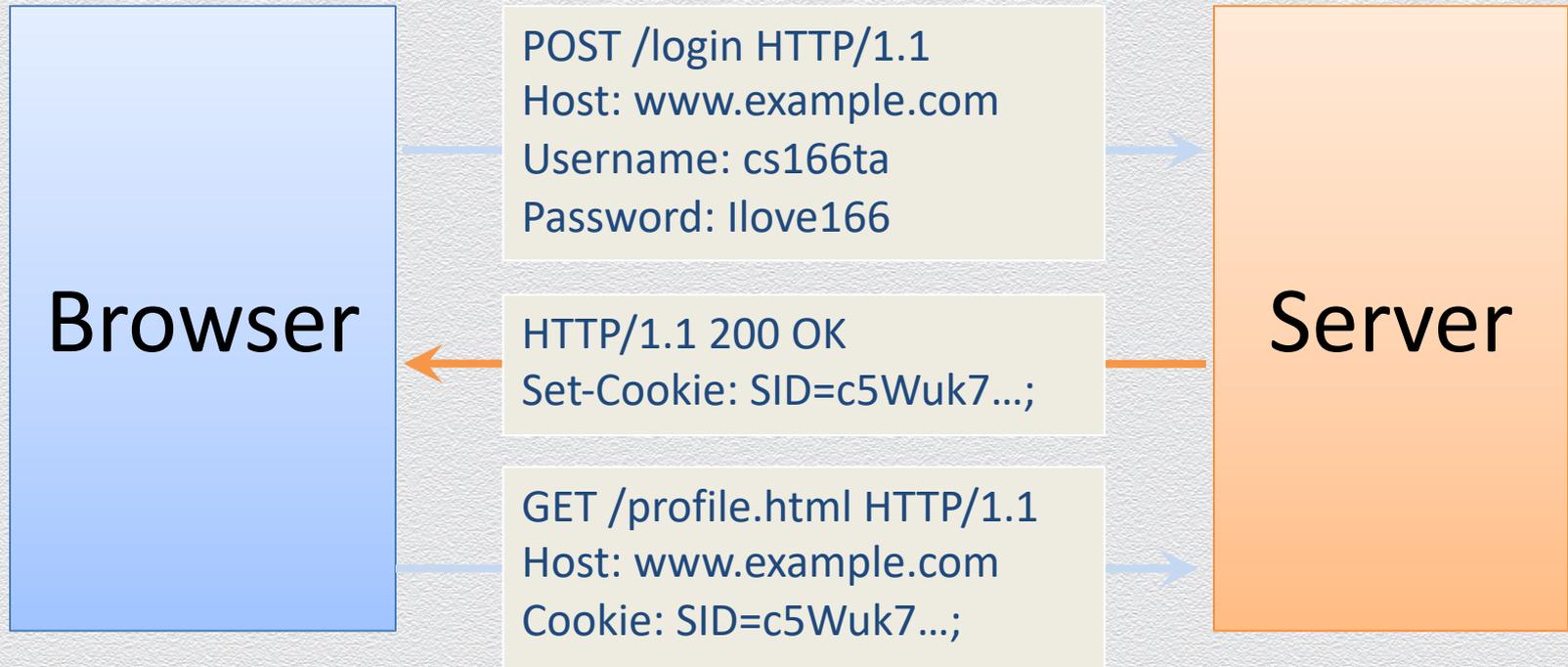
# Specifications for a Session ID

- ◆ Created by server upon successful user authentication
  - ◆ generated as long random string
  - ◆ associated with scope (set of domains) and expiration
  - ◆ sent to browser
- ◆ Kept as secret shared by browser and server
- ◆ Transmitted by browser at each subsequent request to server
  - ◆ must use secure channel between browser and server
- ◆ Session ID becomes invalid after expiration
  - ◆ user asked to authenticate again

# Implementation of Session ID

- ◆ Cookie
  - ◆ Transmitted in HTTP headers
  - ◆ Set-Cookie: SID=c5Wuk7...
  - ◆ Cookie: SID=c5Wuk7...
- ◆ GET variable
  - ◆ Added to URLs in links
  - ◆ <https://www.example.com?SID=c5Wuk7...>
- ◆ POST variable
  - ◆ Navigation via POST requests with hidden variable
  - ◆ `<input type="hidden" name="SID" value="c5Wuk7...">`

# Session ID in Cookie



# Session ID in Cookie (cont.)

- ◆ Advantages
  - ◆ Cookies automatically returned by browser
  - ◆ Cookie attributes provide support for expiration, restriction to secure transmission (HTTPS), and blocking JavaScript access (httponly)
- ◆ Disadvantages
  - ◆ Cookies are shared among all browser tabs
    - ◆ (not other browsers or incognito)
  - ◆ Cookies are returned by browser even when request to server is made from element (e.g., image or form) within page from other server
  - ◆ This may cause browser to send cookies in context not intended by user

# Session ID in GET Variable



# Session ID in GET Variable (cont.)

- ◆ Advantages
  - ◆ Session ID transmitted to server only when intended by user
- ◆ Disadvantages
  - ◆ Session ID inadvertently transmitted when user shares URL
  - ◆ Session ID transmitted to third-party site within referrer
  - ◆ Session ID exposed by bookmarking and logging
  - ◆ Server needs to dynamically generate pages to customize site navigation links and POST actions for each user
  - ◆ Transmission of session ID needs to be restricted to HTTPS on every link and POST action

# Session ID in POST Variable



# Session ID in POST Variable

- ◆ Advantages
  - ◆ Session ID transmitted to server only when intended by user
  - ◆ Session ID not present in URL, hence not logged, bookmarked, or transmitted within referrer
- ◆ Disadvantages
  - ◆ Navigation must be made via POST requests
  - ◆ Server needs to dynamically generate pages to customize forms for each user
  - ◆ Transmission of session ID needs to be restricted to HTTPS on every link and POST action

# OWASP Top Ten (2013-17)

**A1: Injection**

**A2: Broken Authentication and Session Management**

**A3: Cross-Site Scripting (XSS)**

**A4: Broken Access Control**

**A5: Security Misconfiguration**

**A6: Sensitive Data Exposure**

**A7: Insufficient Attack Protection**

**A8: Cross Site Request Forgery (CSRF)**

**A9: Using Components with Known Vulnerabilities**

**A10: Unprotected API**



**OWASP 2013 -2017**



**Just OWASP 2017**



**OWASP**

The Open Web Application Security Project  
<http://www.owasp.org>

# OWASP 2017 - 2021

2017

2021

A01:2017-Injection

A02:2017-Broken Authentication

A03:2017-Sensitive Data Exposure

A04:2017-XML External Entities (XXE)

A05:2017-Broken Access Control

A06:2017-Security Misconfiguration

A07:2017-Cross-Site Scripting (XSS)

A08:2017-Insecure Deserialization

A09:2017-Using Components with Known Vulnerabilities

A10:2017-Insufficient Logging & Monitoring

A01:2021-Broken Access Control

A02:2021-Cryptographic Failures

A03:2021-Injection

(New) A04:2021-Insecure Design

A05:2021-Security Misconfiguration

A06:2021-Vulnerable and Outdated Components

A07:2021-Identification and Authentication Failures

(New) A08:2021-Software and Data Integrity Failures

A09:2021-Security Logging and Monitoring Failures\*

(New) A10:2021-Server-Side Request Forgery (SSRF)\*

\* From the Survey

[www.owasp.org/index.php/Top\\_10](http://www.owasp.org/index.php/Top_10)



**OWASP**

The Open Web Application Security Project  
<http://www.owasp.org>