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

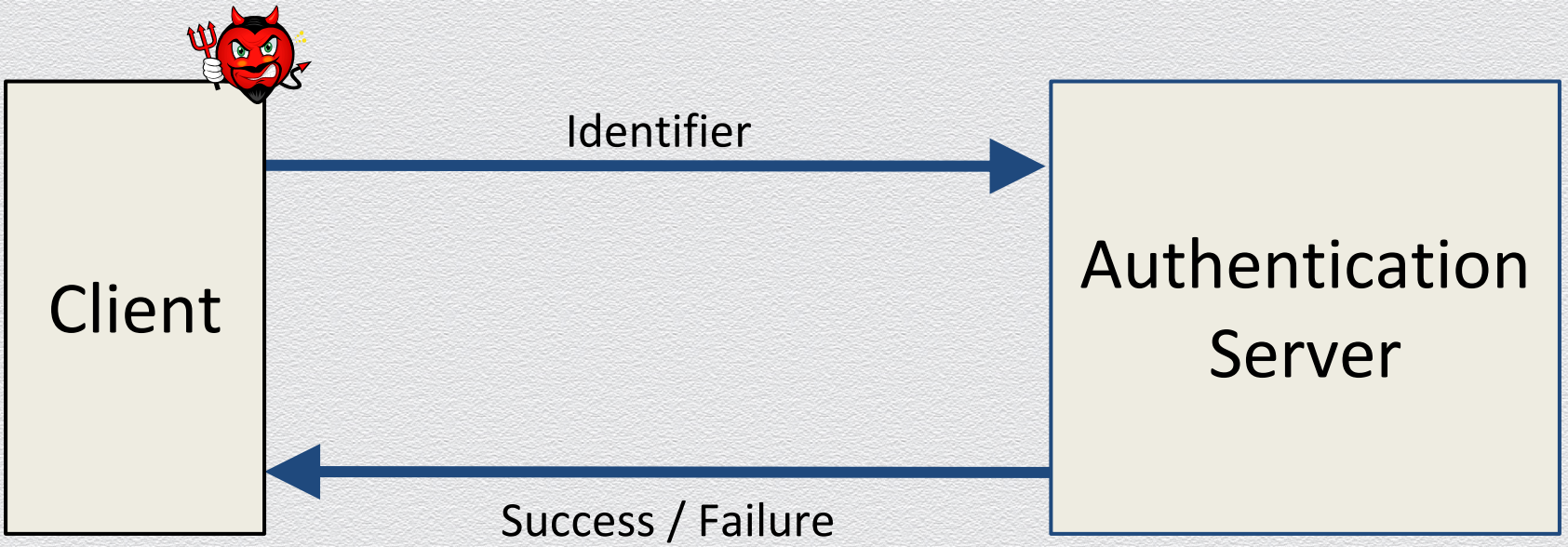
Today

- 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

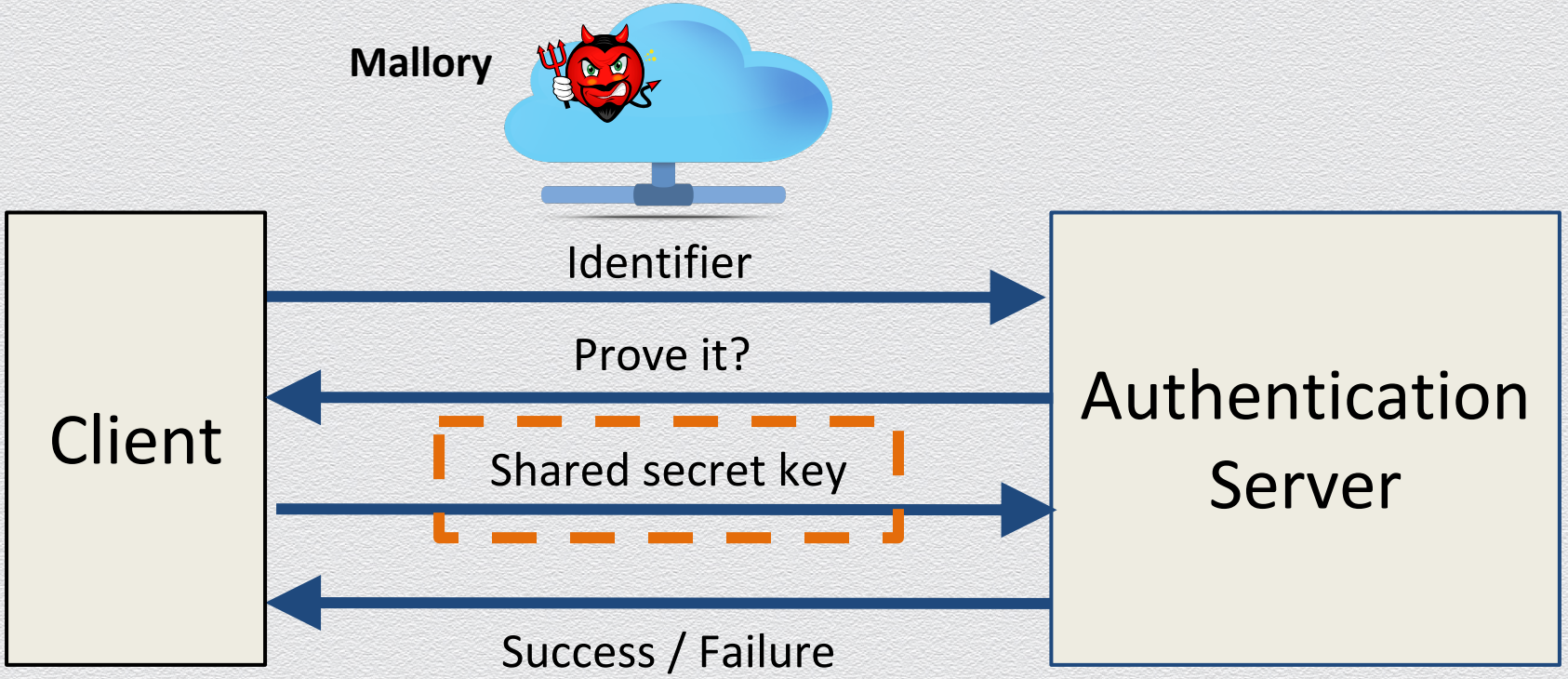
Web security

10.1 Other authentication protocols

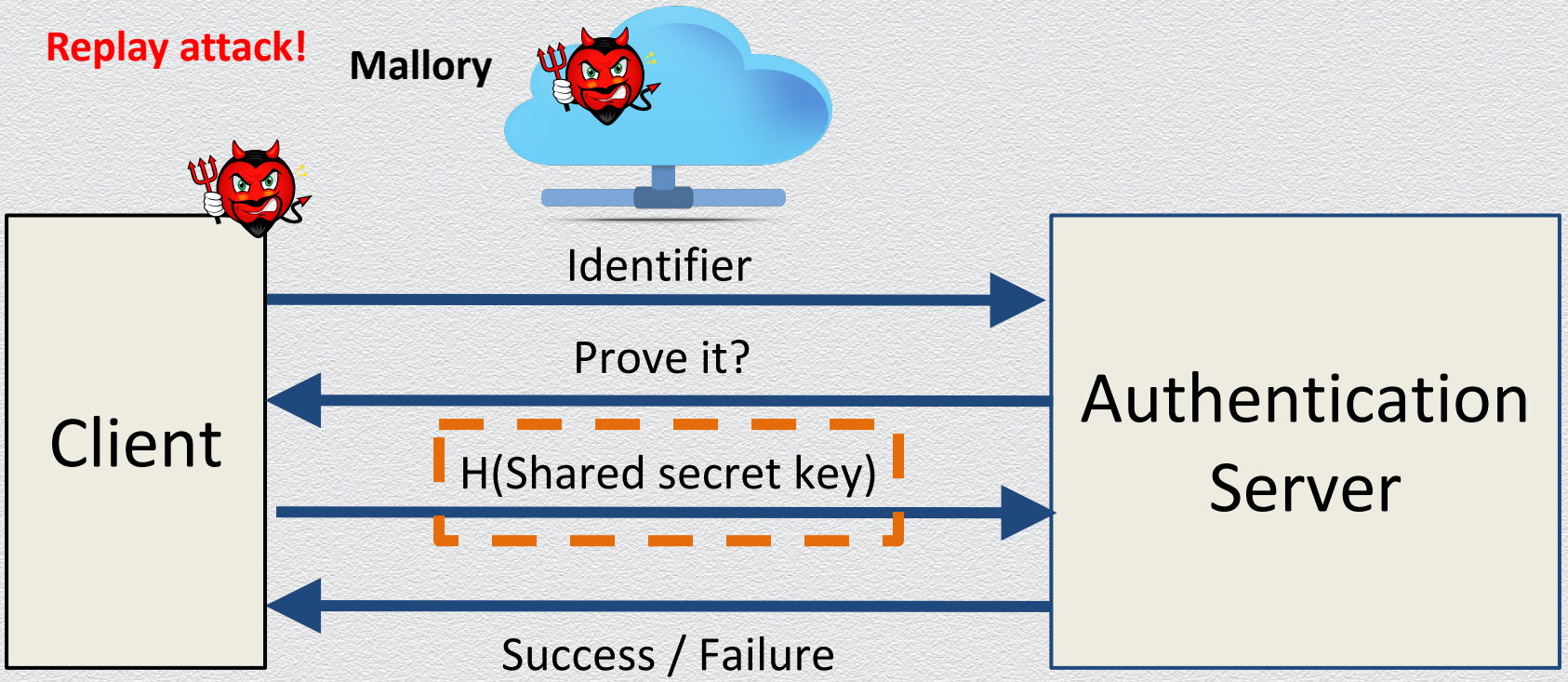
How to authenticate two systems?



But…


Mallory
Identifier
Prove it?
Shared secret key
Success / Failure
Client
Authentication Server

Even better method…



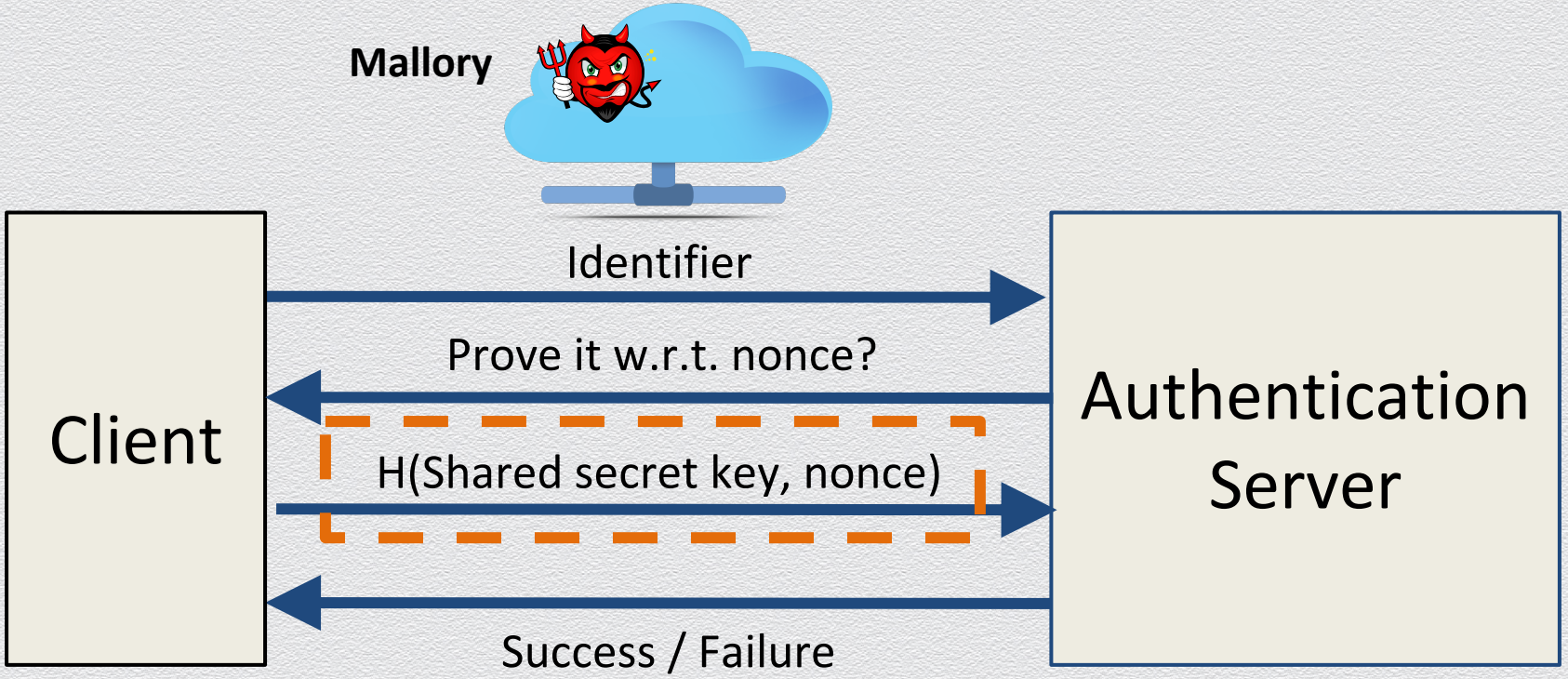
Challenge-response

- 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 attack is not possible
  - Only Client can provide the correct **Response**
  - Server can verify the response

Nonces

- To ensure “freshness”, can employ a **nonce**
  - Nonce == **n**umber 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?

10.2 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

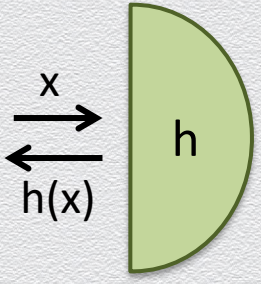
Provable security: Idealized models

- 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



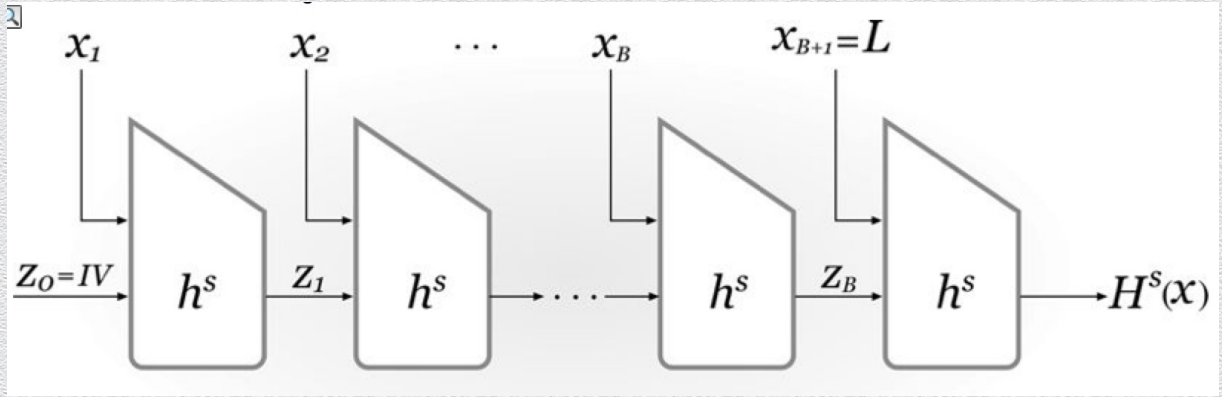
Random-oracle methodology

1. design & analyze using random oracle h; 2. instantiate h with specific function h'

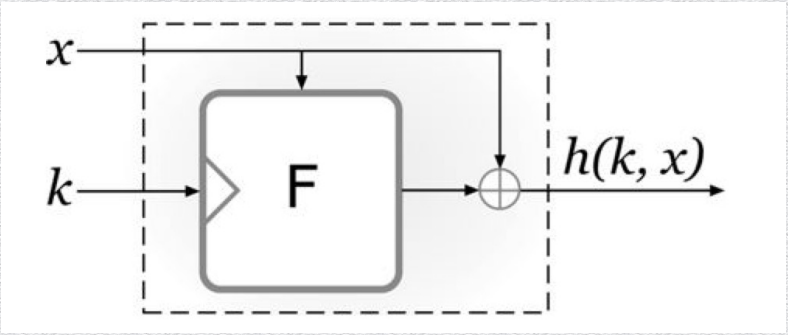
- how sound is such an approach? on-going debate in cryptographic community
- pros (proof in random-oracle model better than no proof at all)
  - leads to significantly **more efficient** (thus practical) schemes
  - design is **sound**, subject to limitations in instantiating h to h'
  - at present, 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

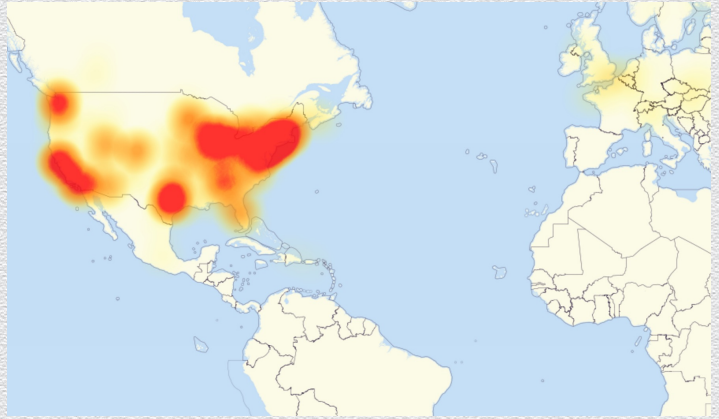


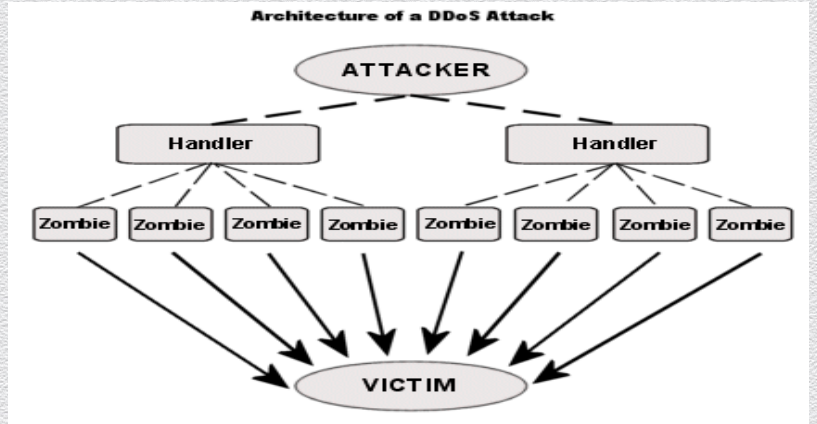
10.3 The Dyn DDoS attack

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

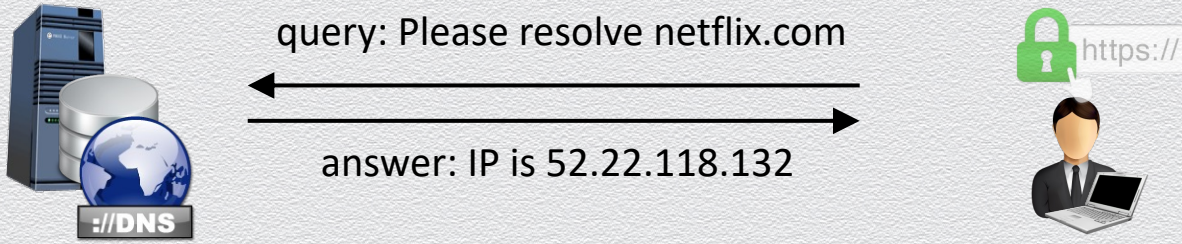




Domain Name Service (DNS) protocol

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



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

netflix.com

52.22.118.132
(or "non-existent")

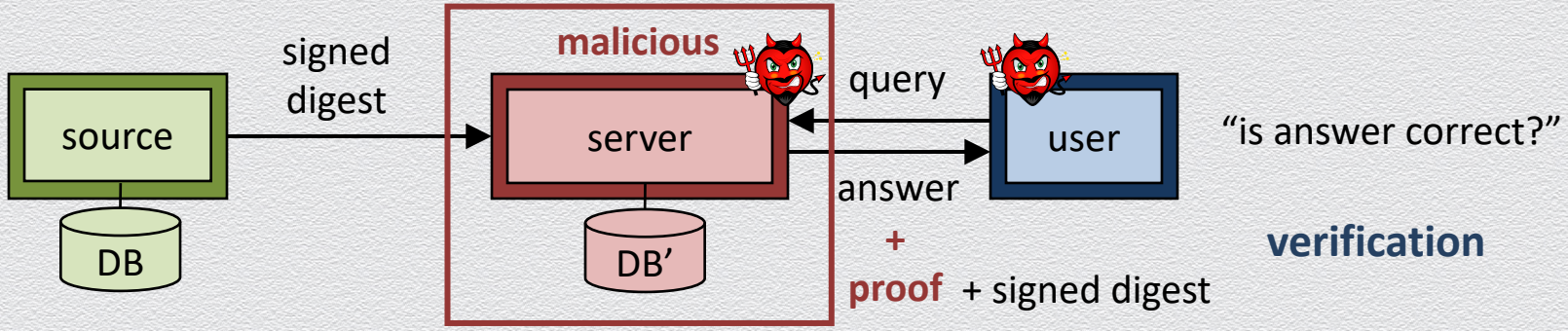
locally cached IP addresses
(at Web browser and OS)

DNS: A critical asset to attack…

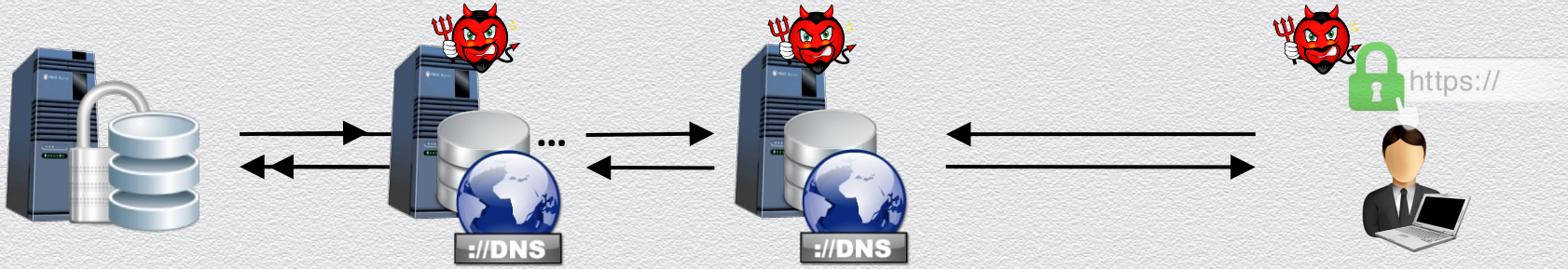
What main security properties must be preserved in such an important service?

- 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

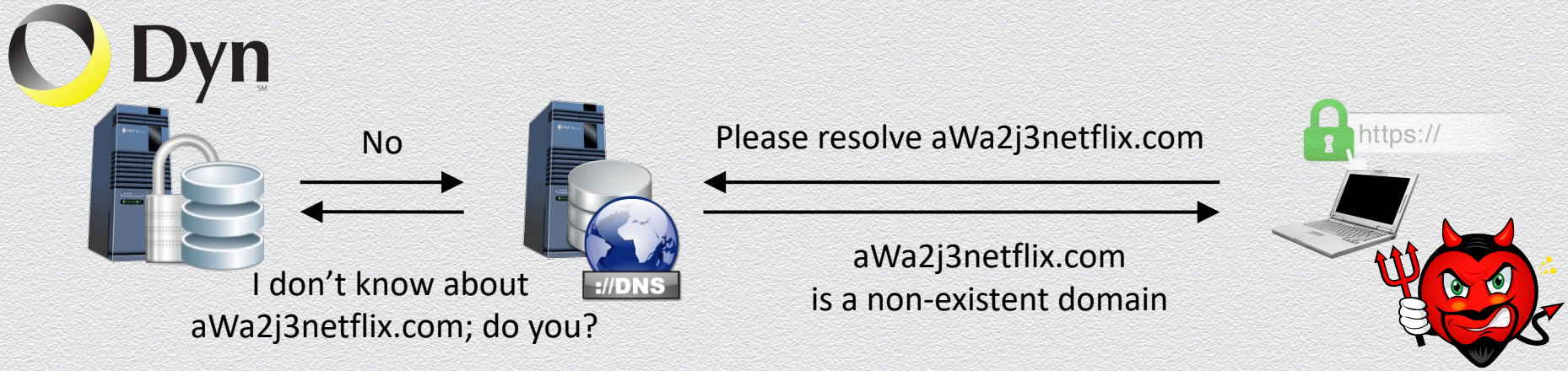
DNS: A critical asset to attack… (cont.)



integrity **availability / confidentiality**



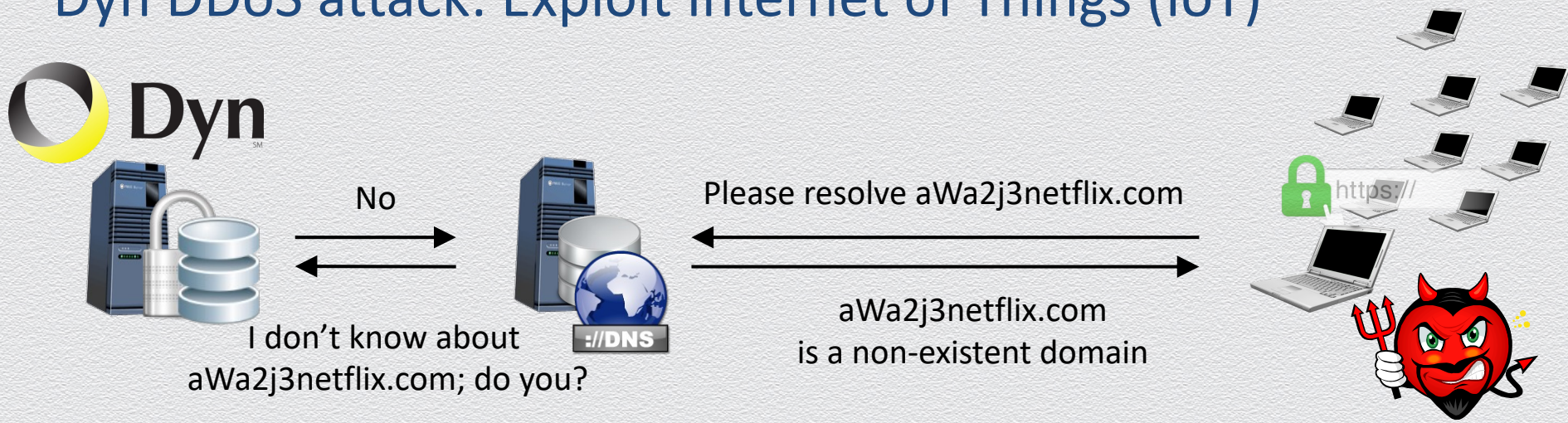
Dyn DDoS attack



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)

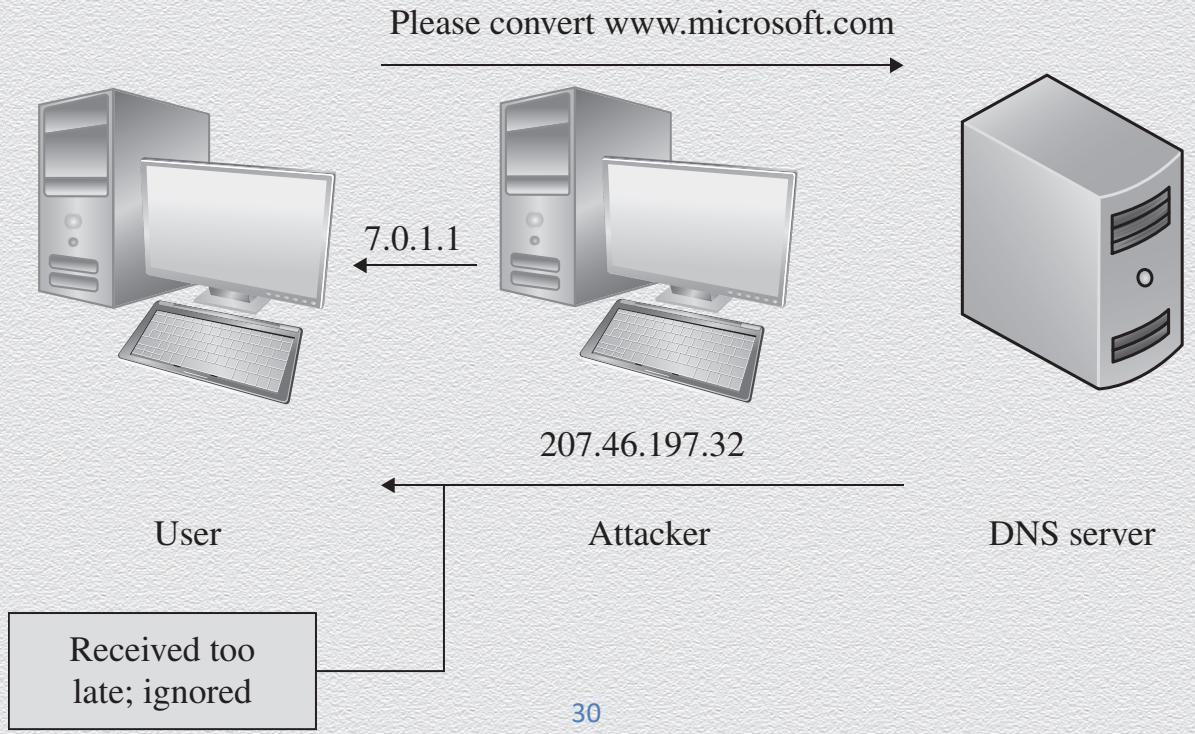


Create a botnet:

- 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...!

DNS spoofing (or cache poisoning)

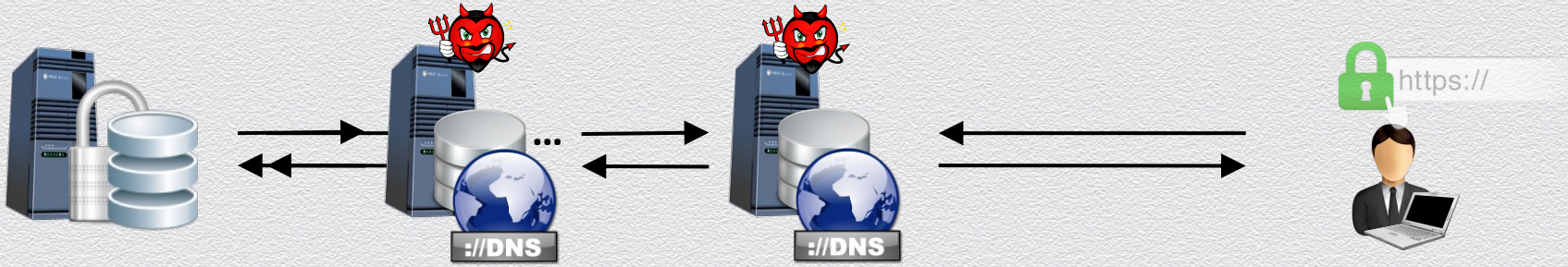
The attacker acts as the DNS server in order to redirect the user to malicious sites



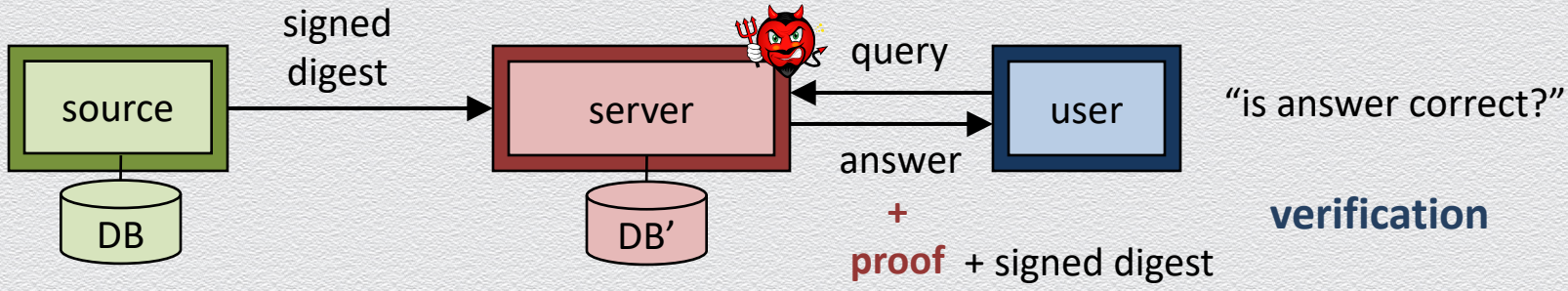
DNSSEC & NSEC

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



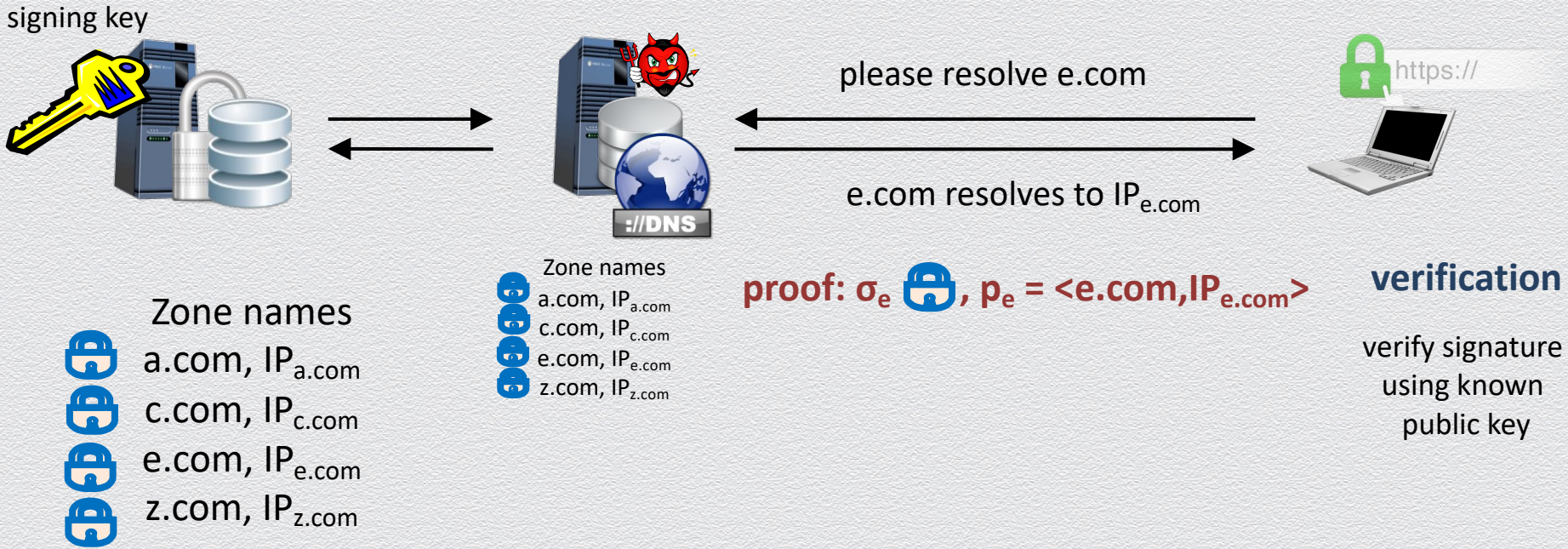
DNSSEC protocol: each DNS entry is pre-signed by primary name server

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>

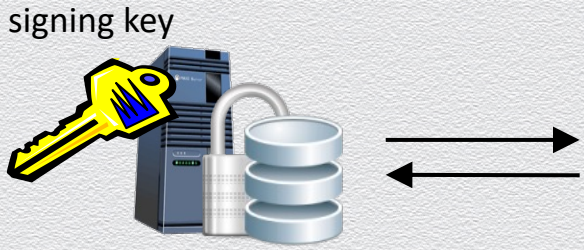
DNSSEC: example

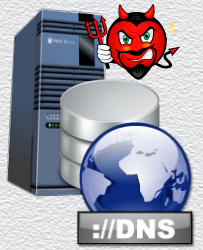
Each entry <domain name, IP address> in the database is individually signed by a primary DNS server and uploaded to secondary DNS servers in signed form



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





please resolve b.com

domain name b.com doesn't exist

proof: σ_1, p_1 = <a.com, c.com>

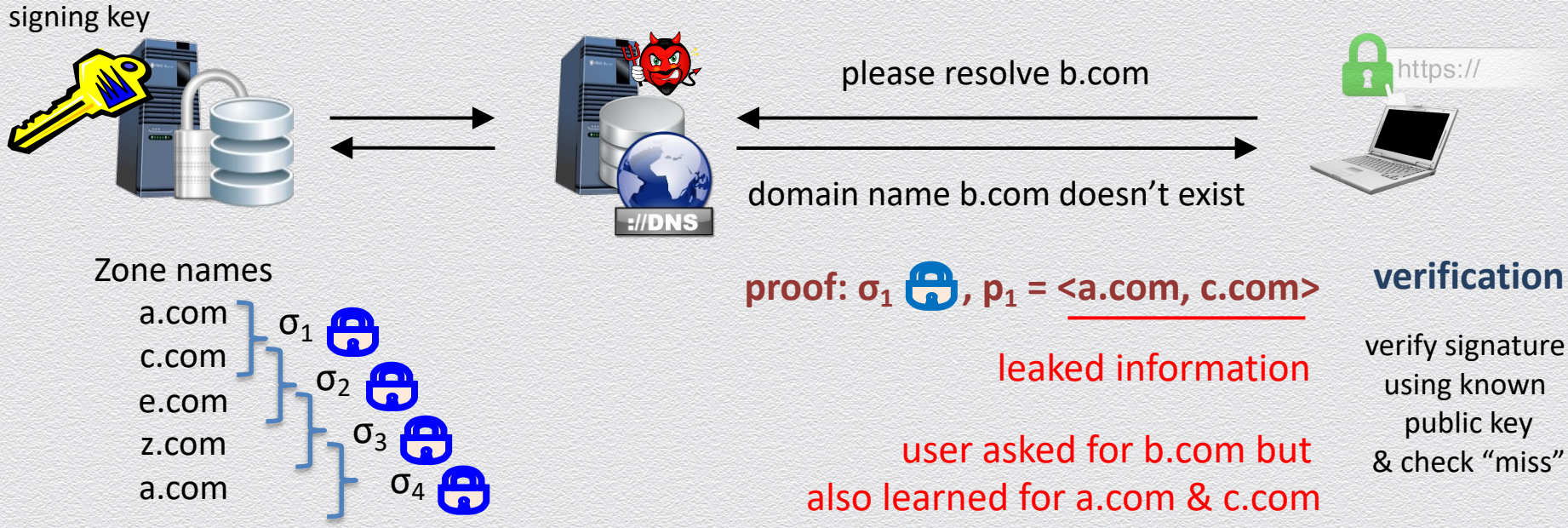
verification

verify signature using known public key & check "miss"

NSEC: Vulnerability

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

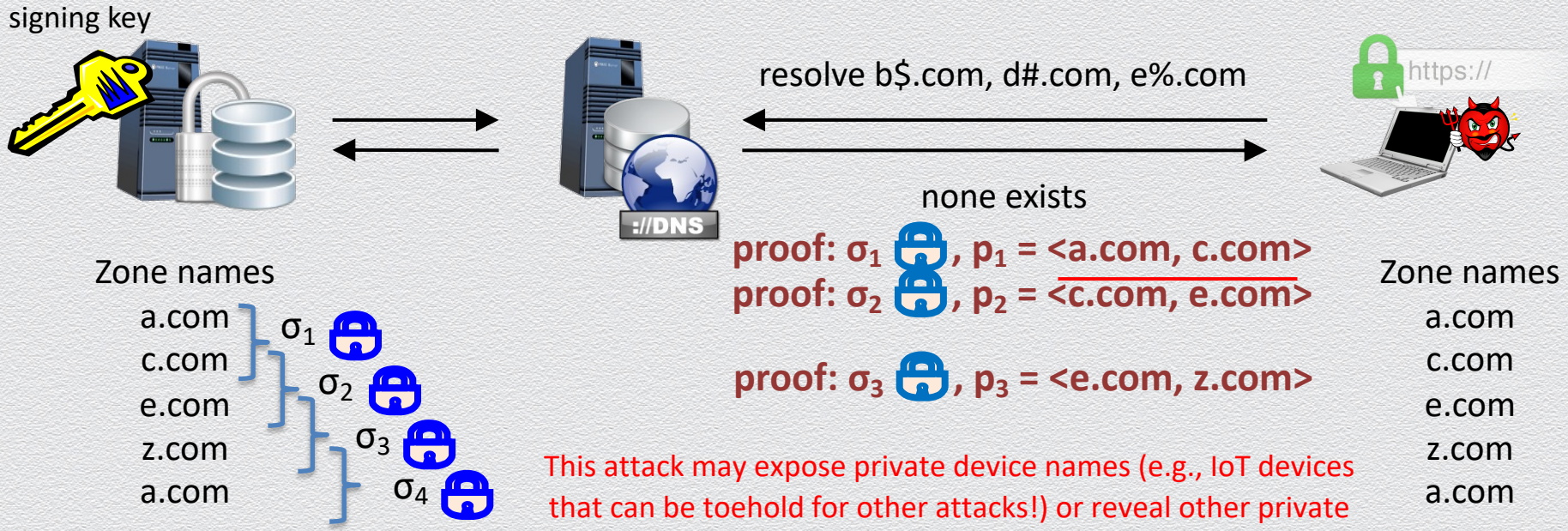
Proofs of non-existing names leak information about other unknown domain names



Zone enumeration attack

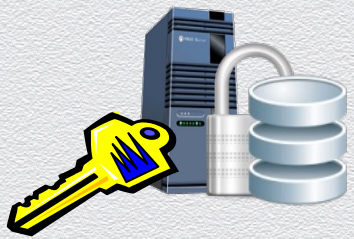
ask for non-existing names
to get all possible proofs

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



This attack may expose private device names (e.g., IoT devices that can be toehold for other attacks!) or reveal other private DNS data that many registries have legal obligations to protect

NSEC3: NSEC in the hash domain



please resolve b.com

b.com is a non-existent domain

proof: σ_3 , p_3 = <dde45,zrit5>

Zone names

a.com
c.com
e.com
z.com

hash h →

a1bb5
23ced
zrit5
dde45

sort →

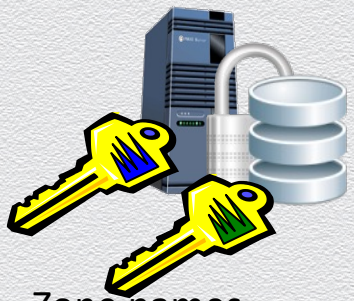
23ced
a1bb5
dde45
zrit5
23ced

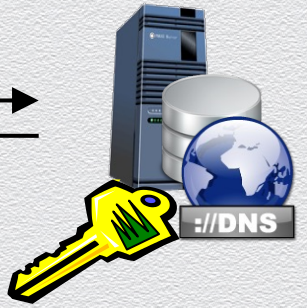
σ_1
σ_2
σ_3
σ_4

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

h(b.com) = ntwo4
e.g., h is SHA-256

NSEC5: A secure solution





please resolve b.com

b.com is a non-existent domain

proof: σ_3 , p_3 = <dde45,zrit5>

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

Zone names

a.com
c.com
e.com
z.com

hash h' →

a1bb5
23ced
zrit5
dde45

sort →

23ced
a1bb5
dde45
zrit5
23ced

σ_1
σ_2
σ_3
σ_4

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

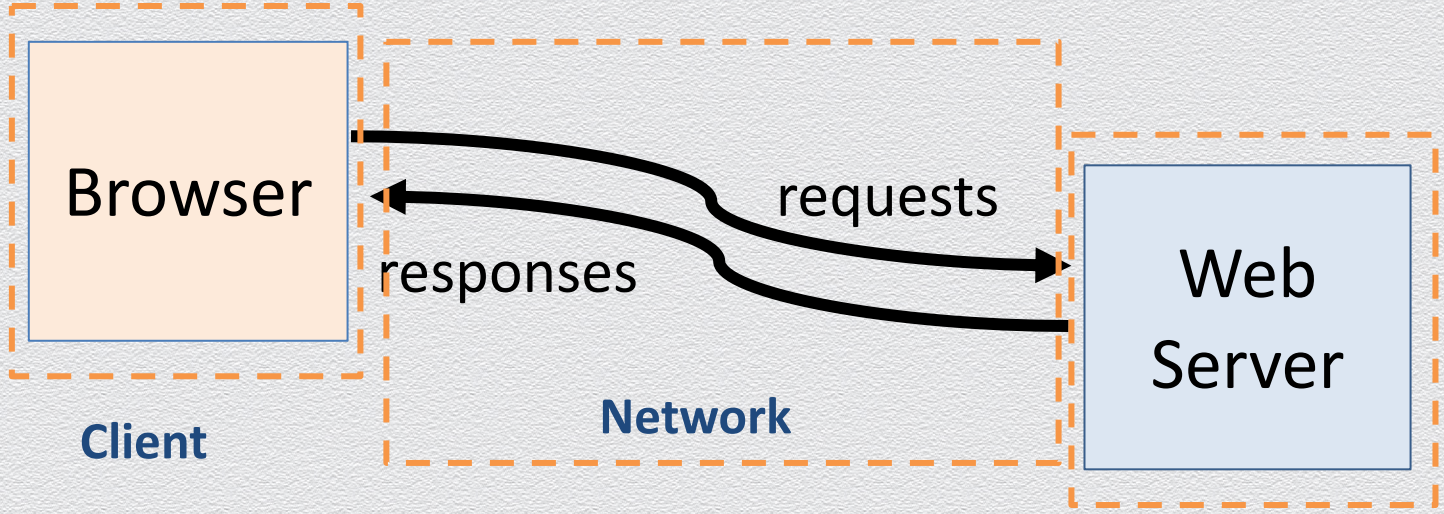
h'(b.com) = ntwo4

h: as in NSEC3
f: "message transformation" hash

h'(x) = h(RSA-Sign( , f(x)))

10.4 Web security model

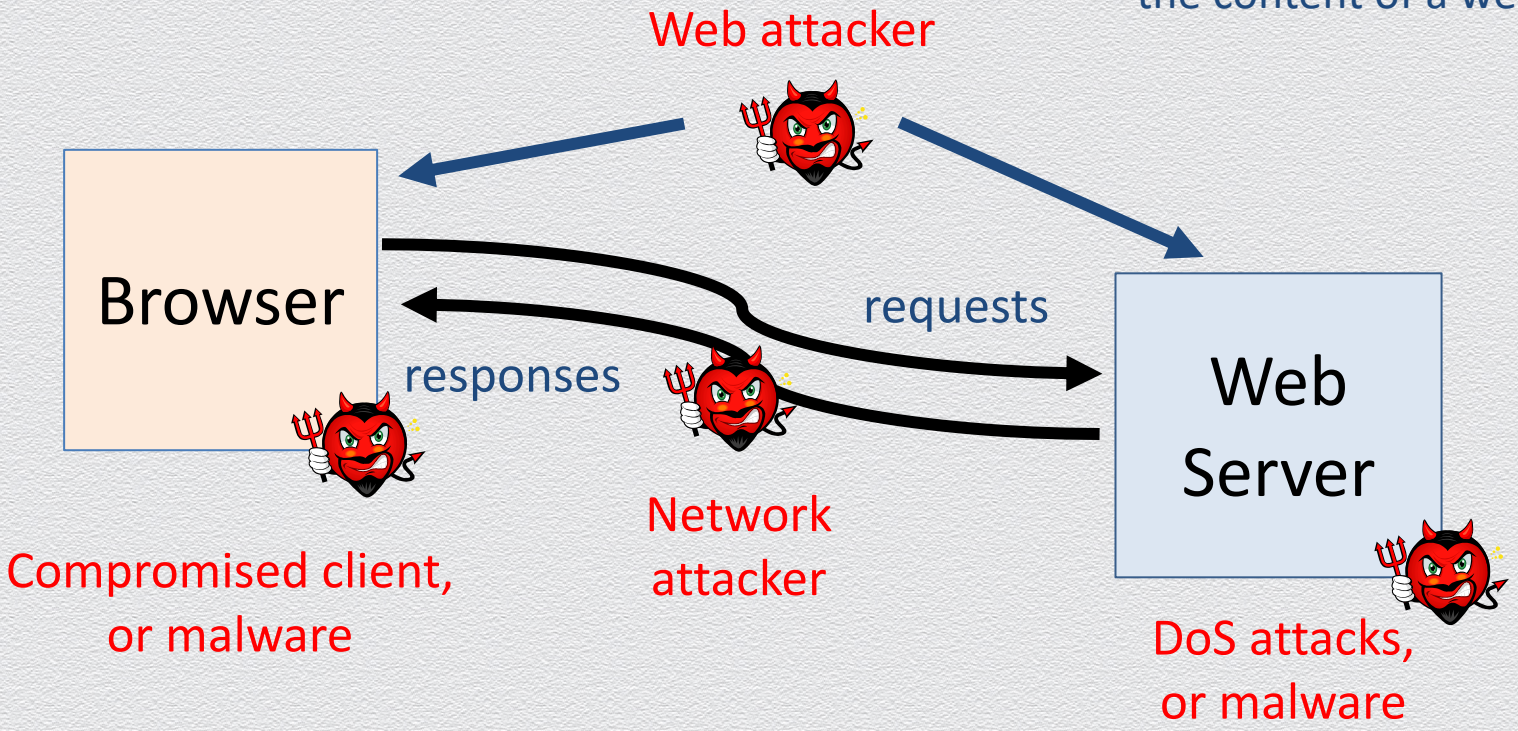
Web applications



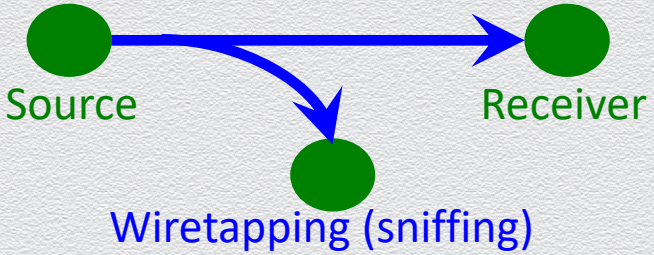
What are the dangers?

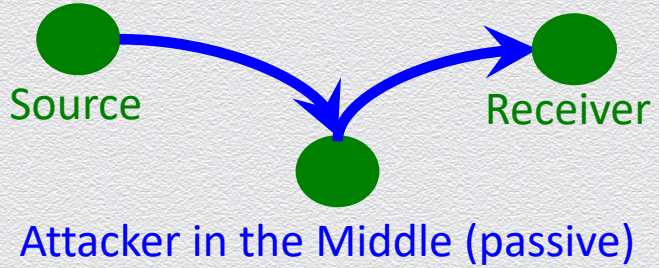
Threat models

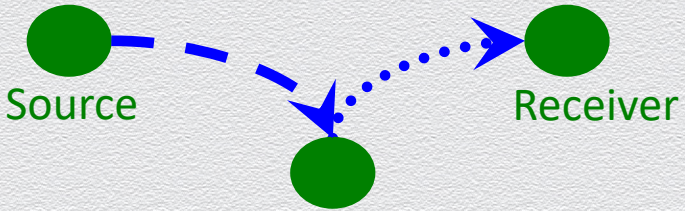
The main vector of attack is via the content of a website

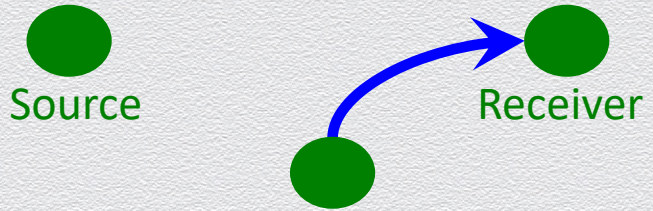


Network attacks









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
  - 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/

We won't address attacks that break the sandbox

- But they happen check the CVE list (Common Vulnerabilities and Exposures)
  - 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

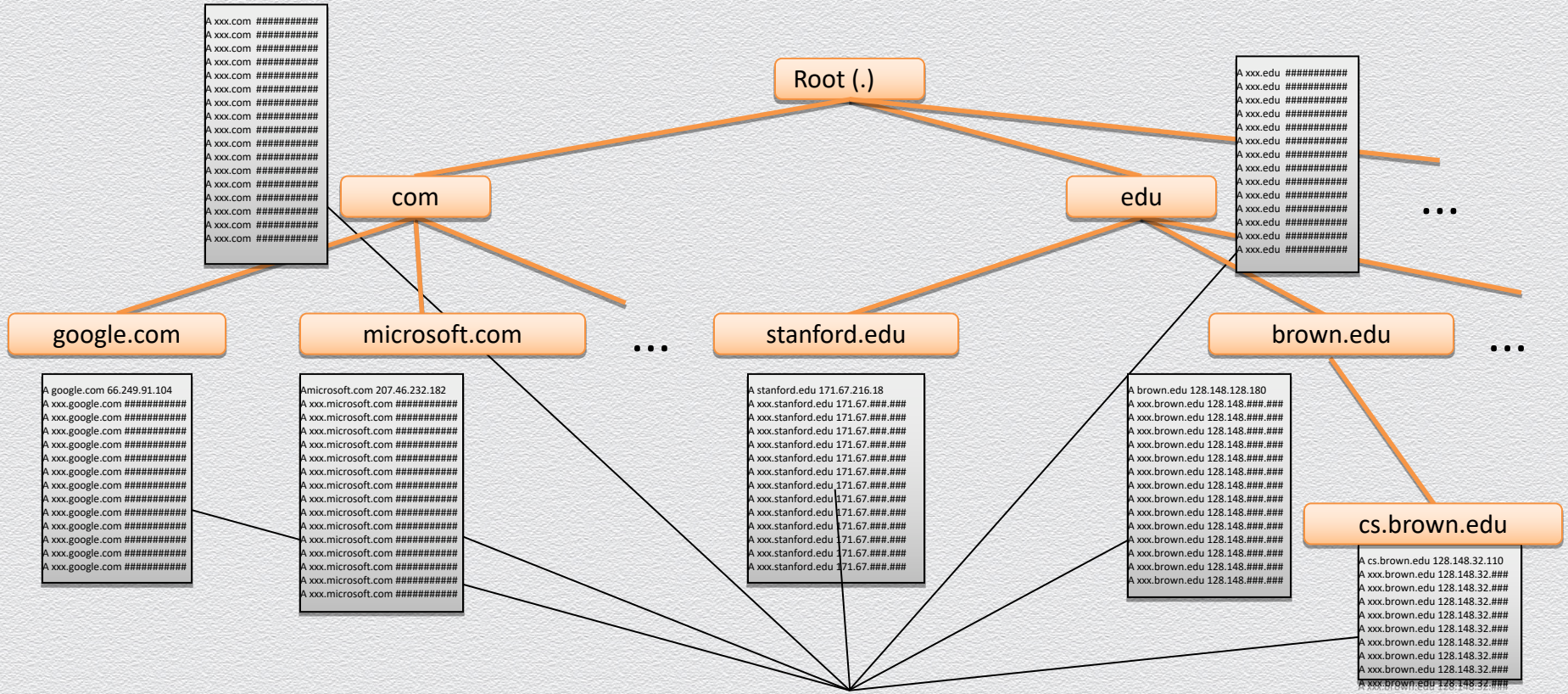
- a protocol
  - e.g. https
- a FQDN
  - e.g. cs.brown.edu
- a path and file name
  - e.g. /about/contacts.html

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



resource records

HTML

Hypertext markup language (HTML)

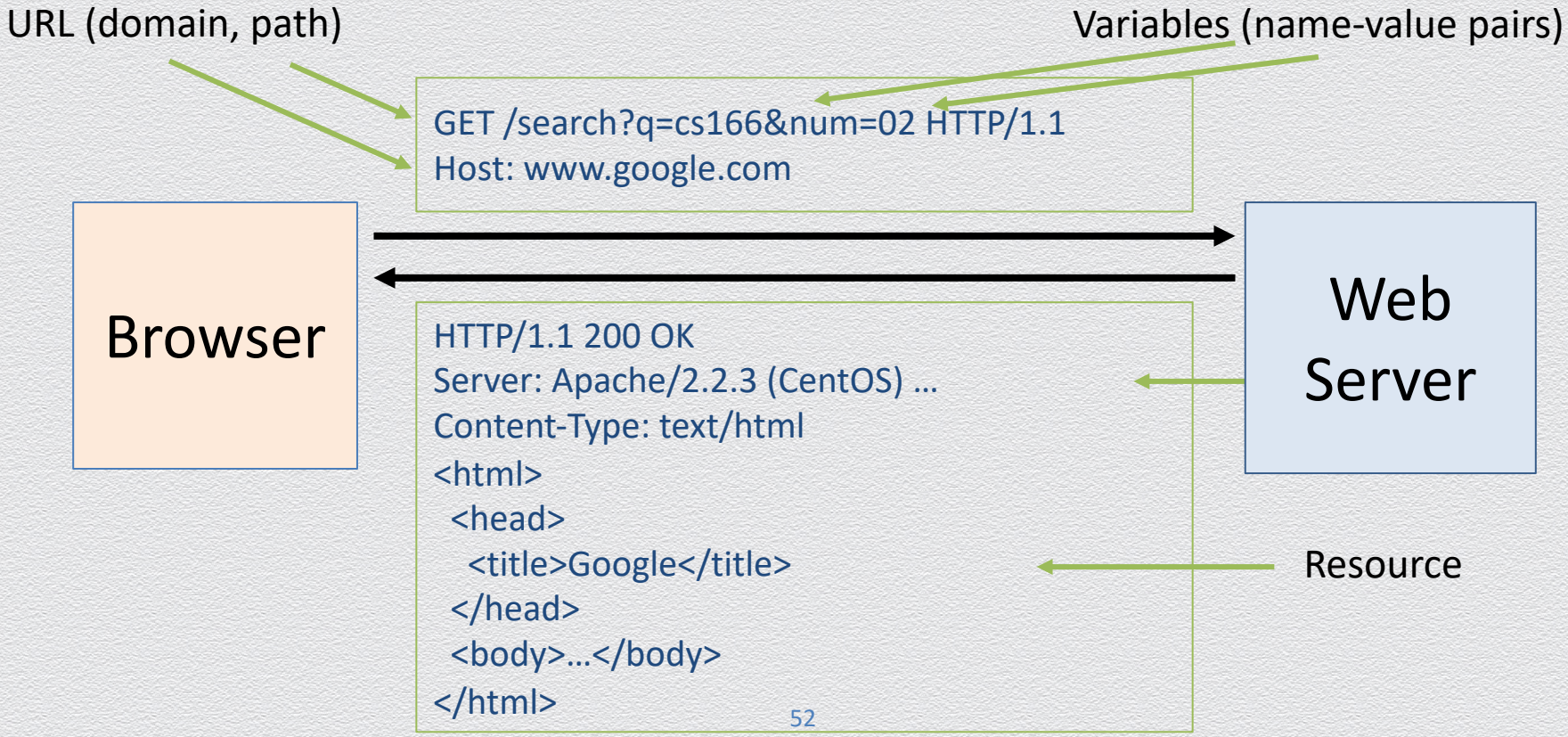
- allows linking to other pages (href)
- supports embedding of images, scripts, other pages (script, iframe)
- user input accepted in forms

```
<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)?



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

	GET	POST
Browser history	✓	X
Browser bookmarking	✓	X
Browser caching	✓	X
Server logs	✓	X
Reloading page	immediate	warning
Variable values	restricted	arbitrary