TLS (continued) and DNS

CS 161 Spring 2022 - Lecture 19

Announcements

- Project 2 is due **Friday, April 8th**
  - There are more design reviews available this week! See Piazza for more details.
  - **Only sign up for a design review if you did not sign up last week!**
- Homework 5 is due **Friday, April 1st**
- Project 1 write-up grades have been released
  - Course staff’s fault they came out so late, sorry!

TLS in Practice

Textbook Chapter 31.3

TLS: Efficiency

- Public-key cryptography: Minor costs
  - Client and server must perform Diffie-Hellman key exchange or RSA encryption/decryption
- Symmetric-key cryptography: Effectively free
  - Modern hardware has dedicated support for symmetric-key cryptography
  - Performance impact is negligible
- Latency: Extra waiting time before the first message
  - Must perform the entire TLS handshake before sending the first message

TLS Provides End-to-End Security

- TLS provides **end-to-end security**: Secure communication between the two endpoints, with no need to trust intermediaries
  - Even if everybody between the client and the server is malicious, TLS provides a secure communication channel
  - End-to-end security does not help if one of the endpoints is malicious (e.g. communicating with a malicious server)
  - Example: An local network attacker (on-path) tries to read our Wi-Fi session, but can't read TLS messages
  - Example: A man-in-the-middle tries to inject TCP packets, but packets will be rejected because the MAC won't be correct
- Using TLS defends against most lower-level network attacks

TLS Does Not Provide Anonymity

- **Anonymity**: Hiding the client's and server's identities from attackers
- An attacker can figure out who is communicating with TLS
  - The certificate is sent during the TLS handshake, containing the server's name
  - The client may also indicate the name of the server in the ClientHello (called Server Name Indication, or SNI)
  - An attacker can see IP addresses and ports of the underlying IP and TCP protocols

TLS Does Not Provide Availability

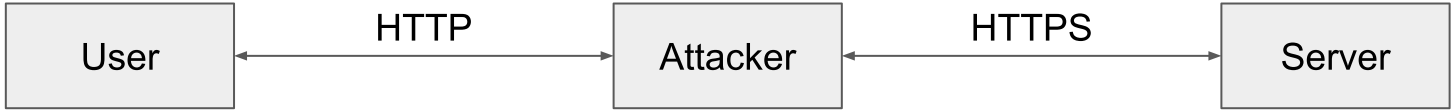
- **Availability**: Keeping the connection open in the face of attackers
- An attacker can stop a TLS connection
  - MITM can drop encrypted TLS packets
  - On-path attacker can still do RST injection to abort the underlying TCP connection
- Result: A TLS connection can still be censored
  - The censor can block TLS connections
  - Networks can also block based on IP addresses
  - TLS 1.3 supports “Encrypted SNI”: The server’s name is requested after the key exchange itself
    - But networks that want to enforce a lack of anonymity (censors, businesses, etc.) can recognize and terminate those connections

TLS for Applications

- Recall Internet layering: TLS provides services to higher layers (the application layer)
- **HTTPS**: The HTTP protocol run over TLS
  - In contrast, HTTP runs over plain TCP, with no TLS added
- Other secure application-layer protocols besides HTTPS exist
  - Pretty much anything that runs over TCP can also run over TLS, since the bytestream abstraction is maintained
  - Example: Email protocol can use the STARTTLS command to uses TLS to secure communications
- TLS does not defend against application-layer vulnerabilities
  - Example: SQL injection, XSS, CSRF, and buffer overflow vulnerabilities in the application are still exploitable over TLS

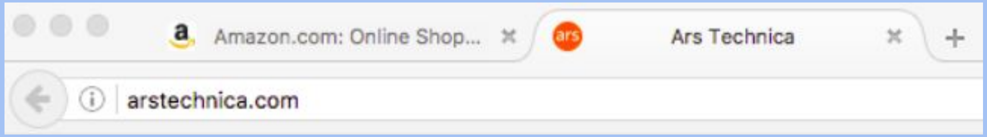
SSL Stripping Attacks

- Browsers often default to using unencrypted HTTP
  - If a user types `google.com` into the browser, the browser opens `http://www.google.com`
  - To mitigate this, websites will often redirect from the HTTP to the HTTPS version of its site
  - This requires the client to first receive the *unprotected* HTTP redirect response
- **SSL stripping**: Forcing a user to use unencrypted HTTP instead of HTTPS
  - A MITM attacker intercepts the first HTTP request and creates their own HTTPS connection to the server
  - The user never receives a redirect to HTTPS, so it believes the site wants them to use HTTP
  - Defense: HTTP Strict-Transport-Security (HSTS) header tells browsers to only access the server with HTTPS

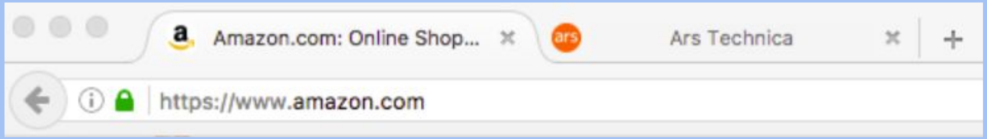


TLS in Browsers

- Original design:
  - When your browser communicates with a server over TLS, your browser displays a lock icon
  - If TLS is not used, there is no lock icon
- What the lock icon means
  - Communication is encrypted (TLS guarantee)
  - You are talking to the legitimate server (TLS guarantee)
  - Any external images or scripts are also fetched over TLS



This website uses HTTP: no lock icon



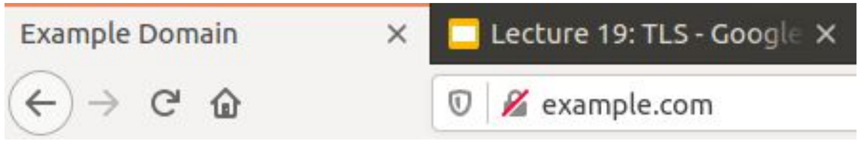
This website uses HTTPS: lock icon

TLS in Browsers

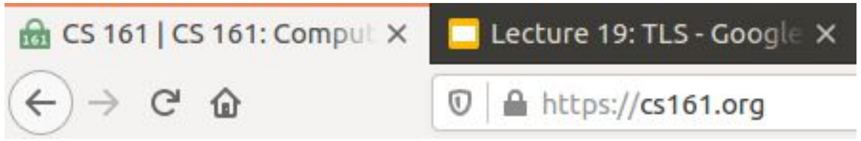
- What users think the lock icon means
  - This website is trustworthy, no matter where the lock icon actually appears
- Attack: The attacker adds their own lock icon somewhere on the page
  - The user thinks they're using TLS, but actually is not using TLS
- Attack: The user might be communicating with an attacker's website over TLS
  - The lock icon appears, but the user is actually vulnerable!

TLS in Browsers

- Modern design: Add a “not secure” icon to connections that don’t use TLS
  - Adds a signal on unencrypted sites
  - Encourages websites to stop supporting all unencrypted, HTTP traffic and redirect to HTTPS



This website uses HTTP: insecure icon



This website uses HTTPS: lock icon

TLS Attack: PRNG Sabotage

- Consider TLS with Diffie-Hellman
  - An attacker who learns the DHE secret *a* can derive the PS g^{ab} mod *p* (recall g^b mod *p* is sent over the channel)
  - An attacker who knows the PS can derive the symmetric keys (recall R_C and R_S are sent over the channel)
- Consider using a PRNG to generate all random values
  - Includes the server DHE secret *a* and the client DHE secret *b*
- What if the PRNG is sabotaged and doesn't have rollback resistance?
  - Example of sabotage: Dual_EC DRBG with knowledge of the secret used to create the generator
  - Example of sabotage: ANSI X9.31: An AES-based PRNG with a secret key
- Attack: See subsequent PRNG output and work backwards to learn the DHE secret

TLS Trust Issues: Certificate Authorities

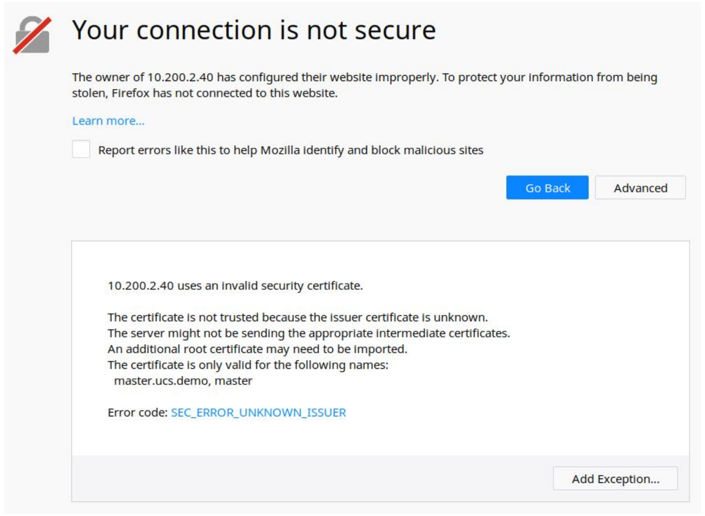
But we can **delegate**…

Recall: Certificates in TLS

- The server sends its certificate
  - Certificate: The server's domain name and public key, signed by a certificate authority
- The browser verifies the server's certificate
  - The browser checks the domain name in the URL matches the domain name in the certificate
  - The certificate authority's public key is hardwired into the browser (trust anchor)
  - The browser uses the CA's public key to verify the signature
- If the certificate is verified, the browser now knows the server's public key

Issues: Unknown Certificate Authority

- What if the browser doesn't have the certificate authority's public key for verification?
- Warn the user that the website is not verified
  - The TLS connection can still proceed, but there is no guarantee that the user is talking to the legitimate server
- What if the user is not talking to the legitimate server?
  - No more end-to-end security
  - An attacker can read and modify messages
  - An attacker can impersonate the server


Your connection is not secure

The owner of 10.200.2.40 has configured their website improperly. To protect your information from being stolen, Firefox has not connected to this website.

Learn more...

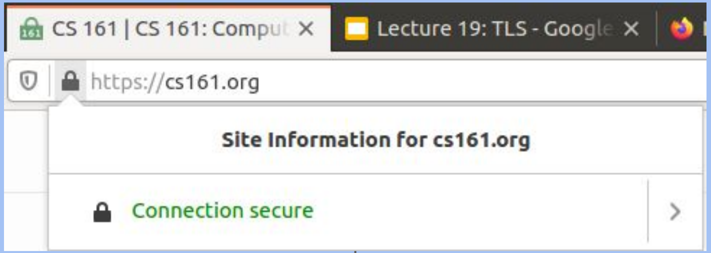
Report errors like this to help Mozilla identify and block malicious sites

10.200.2.40 uses an invalid security certificate.

The certificate is not trusted because the issuer certificate is unknown.
The server might not be sending the appropriate intermediate certificates.
An additional root certificate may need to be imported.
The certificate is only valid for the following names:
master.ucs.demo, master

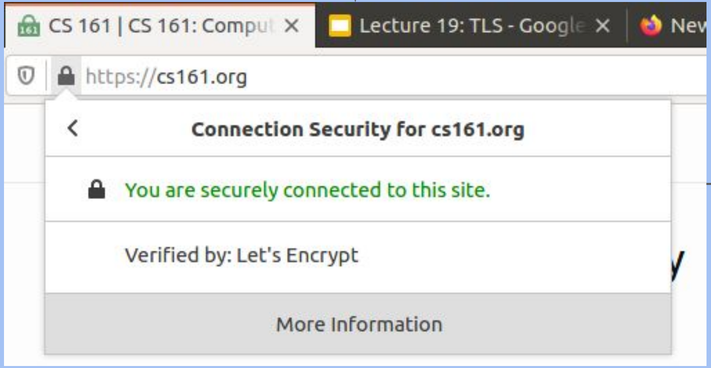
Error code: SEC_ERROR_UNKNOWN_ISSUER

Verifying Certificates


CS 161 | CS 161: Comput
Lecture 19: TLS - Google
https://cs161.org

Site Information for cs161.org

Connection secure

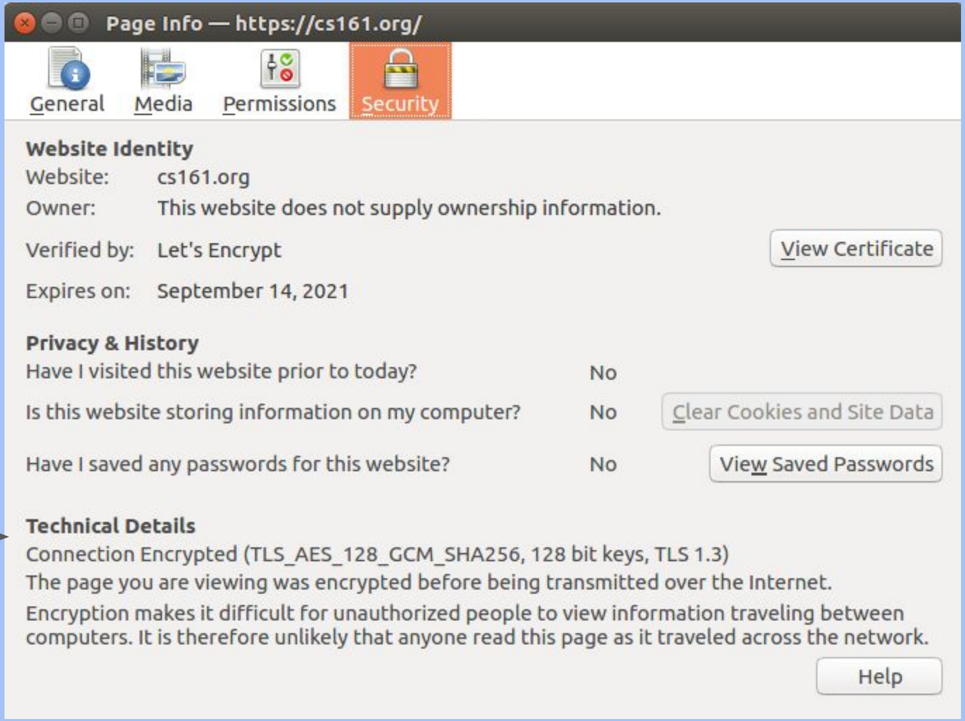

CS 161 | CS 161: Comput
Lecture 19: TLS - Google
https://cs161.org

Connection Security for cs161.org

You are securely connected to this site.

Verified by: Let's Encrypt

More Information


Page Info — https://cs161.org/

General | Media | Permissions | Security

Website Identity

Website:	cs161.org
Owner:	This website does not supply ownership information.
Verified by:	Let's Encrypt
Expires on:	September 14, 2021

View Certificate

Privacy & History

Have I visited this website prior to today?	No
Is this website storing information on my computer?	No
Have I saved any passwords for this website?	No

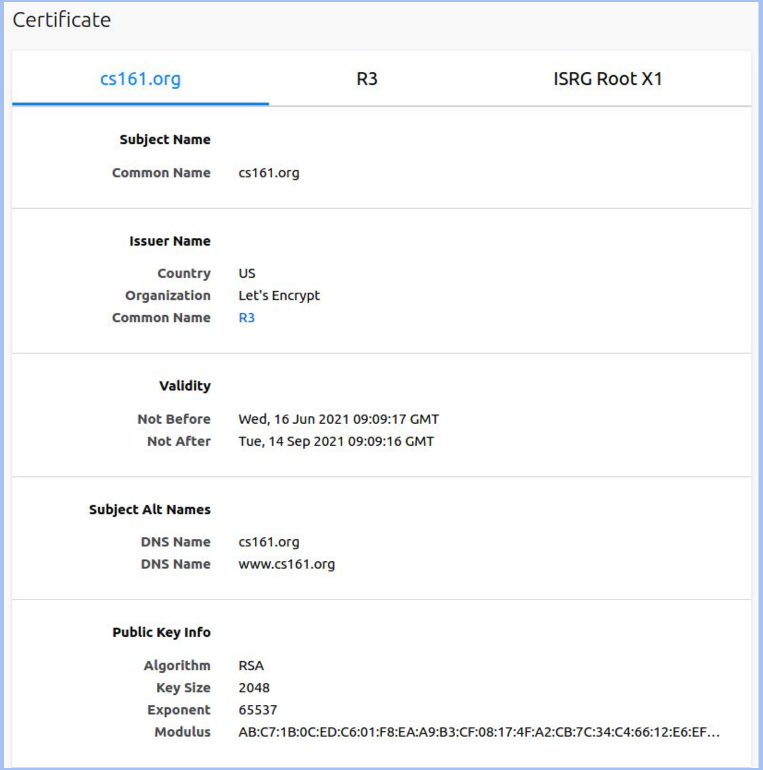
Clear Cookies and Site Data
View Saved Passwords

Technical Details

Connection Encrypted (TLS_AES_128_GCM_SHA256, 128 bit keys, TLS 1.3)
The page you are viewing was encrypted before being transmitted over the Internet.
Encryption makes it difficult for unauthorized people to view information traveling between computers. It is therefore unlikely that anyone read this page as it traveled across the network.

Help

Verifying Certificates


Certificate

cs161.org | R3 | ISRG Root X1

Subject Name	
Common Name	cs161.org
Issuer Name	
Country	US
Organization	Let's Encrypt
Common Name	R3
Validity	
Not Before	Wed, 16 Jun 2021 09:09:17 GMT
Not After	Tue, 14 Sep 2021 09:09:16 GMT
Subject Alt Names	
DNS Name	cs161.org
DNS Name	www.cs161.org
Public Key Info	
Algorithm	RSA
Key Size	2048
Exponent	65537
Modulus	AB:C7:1B:0C:ED:C6:01:F8:EA:A9:B3:CF:08:17:4F:A2:CB:7C:34:C4:66:12:E6:EF…

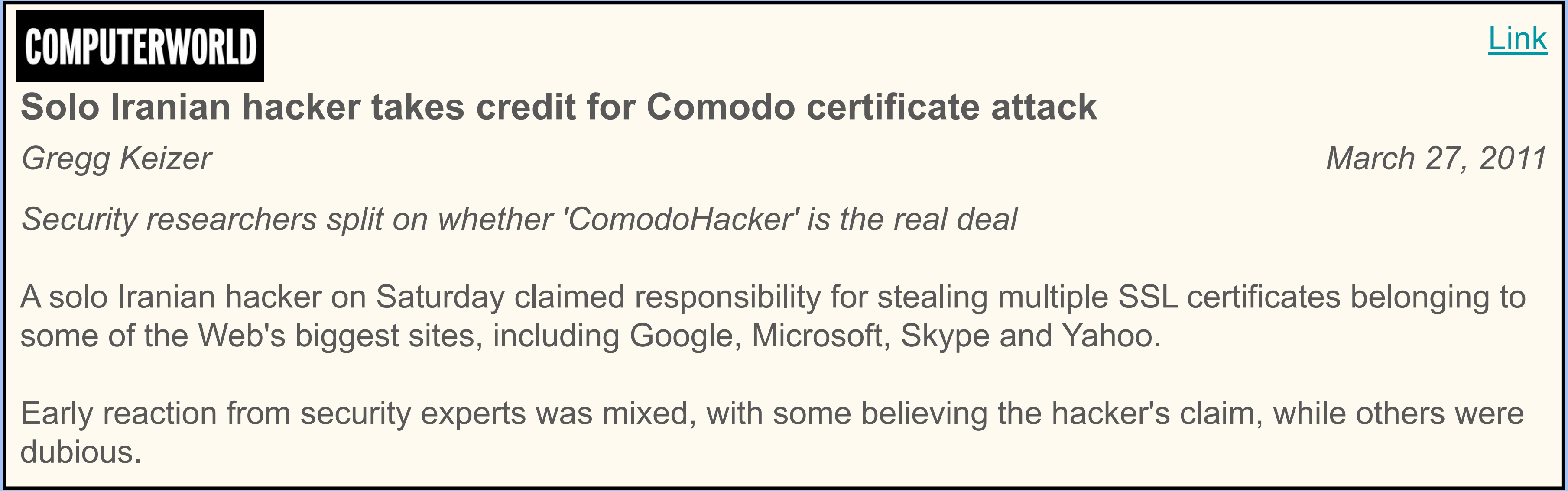
Issues: Revocation

- What if an attacker steals a server's private key?
  - The certificate with the corresponding public key is no longer valid
  - TLS certificates have an expiration date, but they often don't expire for years
- Solution: Certificate revocation lists
  - The CA occasionally sends out lists of certificates that are no longer valid
  - The browser occasionally downloads the lists
- Solution: Online Certificate Status Protocol (OCSP)
  - The browser queries the CA whether a given certificate is still valid
  - The CA responds either "good" or "revoked," signed with the CA's private key

Issues: Trust Anchors

- How many certificate authorities do we need to implicitly trust?
  - Modern browsers implicitly trust 100–200 root certificate authorities
- A CA might issue a malicious certificate (e.g. stating that attacker’s public key belongs to Google) because:
  - The CA is hacked
  - An attacker pays the CA to issue a malicious certificate

Issues: Trust Anchors



COMPUTERWORLD

Link

Solo Iranian hacker takes credit for Comodo certificate attack

Gregg Keizer *March 27, 2011*

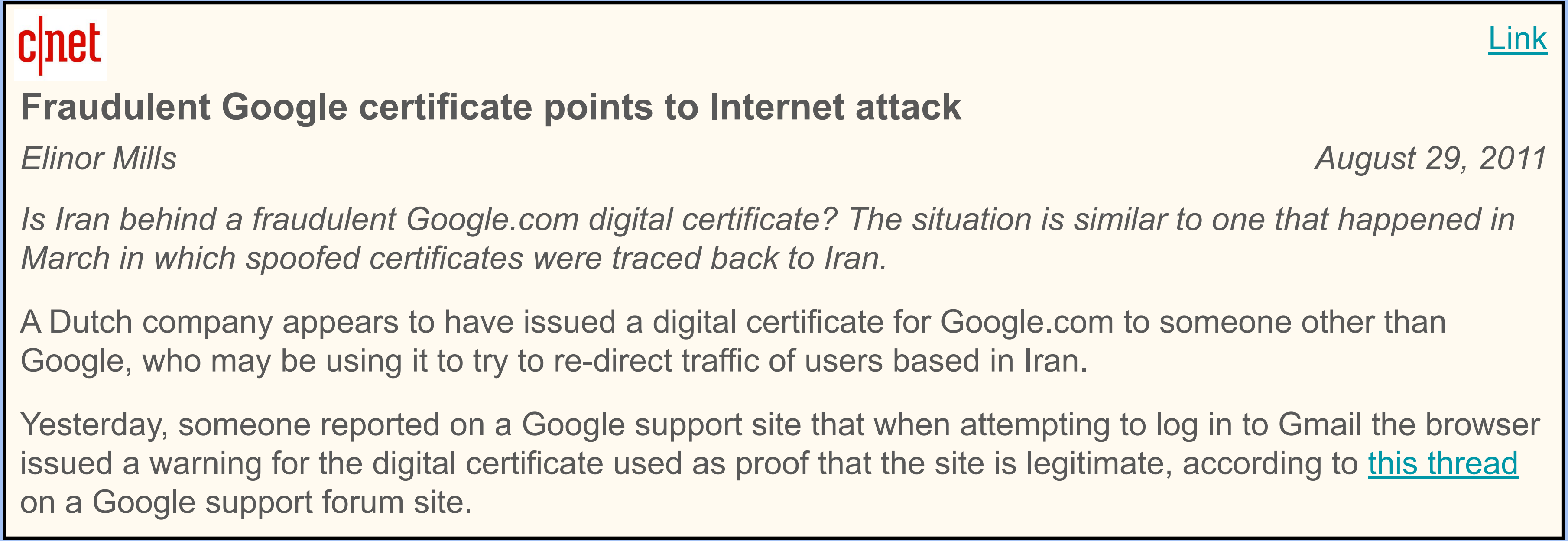
Security researchers split on whether 'ComodoHacker' is the real deal

A solo Iranian hacker on Saturday claimed responsibility for stealing multiple SSL certificates belonging to some of the Web's biggest sites, including Google, Microsoft, Skype and Yahoo.

Early reaction from security experts was mixed, with some believing the hacker's claim, while others were dubious.

Takeaway: Trust certificate authorities can be compromised by hackers

Issues: Trust Anchors



c|net

Link

Fraudulent Google certificate points to Internet attack

Elinor Mills

August 29, 2011

Is Iran behind a fraudulent Google.com digital certificate? The situation is similar to one that happened in March in which spoofed certificates were traced back to Iran.

A Dutch company appears to have issued a digital certificate for Google.com to someone other than Google, who may be using it to try to re-direct traffic of users based in Iran.

Yesterday, someone reported on a Google support site that when attempting to log in to Gmail the browser issued a warning for the digital certificate used as proof that the site is legitimate, according to this thread on a Google support forum site.

Takeaway: Trust certificate authorities can be compromised by hackers

Issues: Trust Anchors

threatpost [Link]

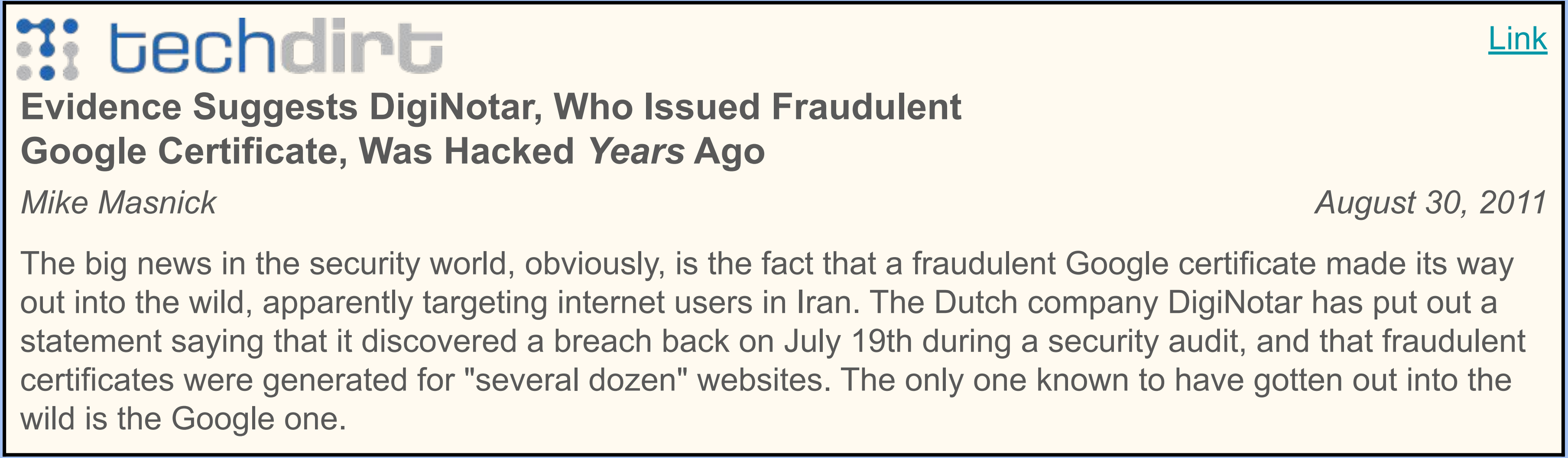
Final Report on DigiNotar Hack Shows Total Compromise of CA Servers

Dennis Fisher *October 31, 2012*

The attacker who penetrated the Dutch CA DigiNotar last year had complete control of all eight of the company's certificate-issuing servers during the operation and he may also have issued some rogue certificates that have not yet been identified.

Takeaway: Trust certificate authorities can be compromised by hackers

Issues: Trust Anchors



techdirt

Link

Evidence Suggests DigiNotar, Who Issued Fraudulent Google Certificate, Was Hacked *Years* Ago

Mike Masnick *August 30, 2011*

The big news in the security world, obviously, is the fact that a fraudulent Google certificate made its way out into the wild, apparently targeting internet users in Iran. The Dutch company DigiNotar has put out a statement saying that it discovered a breach back on July 19th during a security audit, and that fraudulent certificates were generated for "several dozen" websites. The only one known to have gotten out into the wild is the Google one.

Takeaway: Trust certificate authorities can be compromised by hackers

Issues: Trust Anchors

- DigiNotar: A certificate authority that was hacked
  - All web browsers removed DigiNotar from the list of trusted CAs
- WoSign: An untrustworthy certificate authority
  - Also removed by all browsers
  - A user who controls `nweaver.github.com` can create certificates for any subdomain of `github.com`
- **Takeaway**: It is hard to implicitly trust the root CAs (trust anchors) in TLS
- **Takeaway**: TLS CA ecosystem is largely secured through human relationships
  - CAs are motivated to be good because browser vendors can pronounce a “death sentence” on bad certificate authorities at any time (removing them from the list of trusted root CAs)

Solving Trust Issues

- **Certificate pinning**: The browser restricts which CAs are allowed to issue a certificate for each website
    - Example: Only the Google CA is allowed to sign certificates for Google websites
    - Now creating a fake certificate for a specific website requires attacking a particular CA
- **Certificate transparency**: Public logs provided by CAs
    - Specifics are out of scope
    - High-level idea: Use hash chains to keep a record of all issued certificates
    - The server can tell the browser to only accept certificates from CAs implementing transparency
    - The server can then make sure that a CA never screws up

Solving Trust Issues

- Other solutions implementing to “trust but verify” a certificate you received
  - **EFF’s SSL Observatory**: Check against certificates seen by other dedicated computers, called “observatories,” placed around the Internet
  - **ICSI’s Certificate Notary**: Check against certificates used in common Internet traffic, by tapping into common Internet channels

Certificate Authority Example: Let’s Encrypt

- TLS requires every website to obtain and maintain certificates
    - Cost overhead: Certificates might cost money
    - Some management overhead involved
- **Let’s Encrypt**: The world’s largest certificate authority
    - Issues certificates for free
    - Tries to make obtaining certificates as easy as possible:
      Uses the ACME protocol to validate sites
    - Certificates only last for a short period of time (*forcing sites to automate renewing them*)
- Steps of issuing a certificate (can all be automated with a script)
    - The server requests a certificate
    - Let’s Encrypt gives the server a file and tells the server to upload the file at a specific location
    - The server uploads the file to the website
    - Let’s Encrypt verifies that the file has appeared on the website (thus verifying the server’s identity) and issues the certificate to the server

Using Let’s Encrypt in Go

- It is literally just a 2-liner to use Let's Encrypt with Go to turn an HTTP Go server into HTTPS:

```
import "golang.org/x/crypto/acme/autocert"

...
   log.Fatal(http.Serve(autocert.NewListener("example.com"), handler))
```

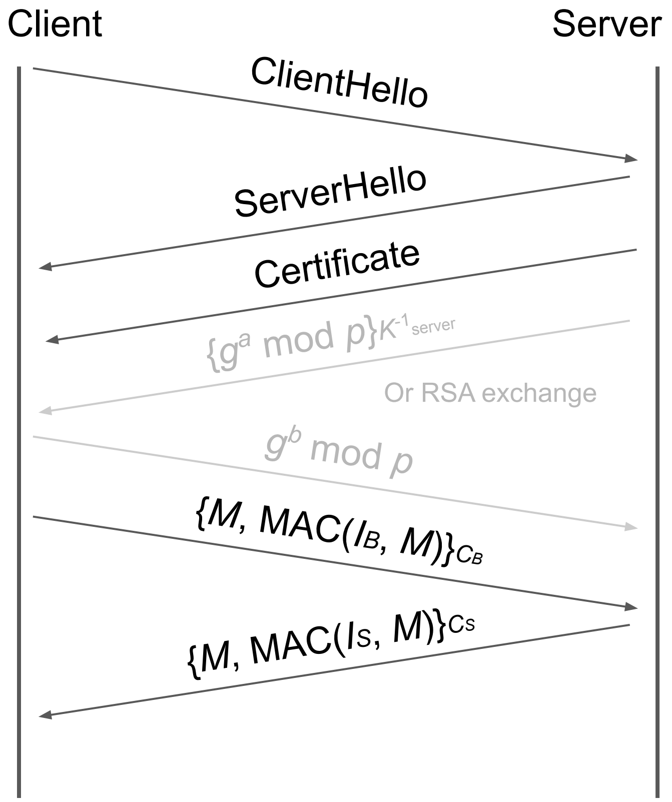
- **Takeaway**: Enabling HTTPS in web servers today is often very simple!

TLS Today

- Between Let’s Encrypt and browser changes, TLS is no longer something special
  - It used to require a fair bit of effort and non-trivial money to set up!
- Instead, TLS is the default
- Biggest limitation with HTTPS:
  - A page loaded over HTTPS should *only* include content from other HTTPS pages: no items from HTTP pages
  - Advertisement networks were the biggest bottleneck in transitioning the web to almost universal support of HTTPS

TLS: Summary

- TLS Handshake
  - Nonces make every handshake different (prevents replay attacks across connections)
  - Certificate proves server's public key
  - RSA or DHE proves that the server owns the private key
  - RSA or DHE helps client and server agree on a shared secret key
  - MAC exchange ensures no one tampered with the handshake
  - Messages are sent with symmetric encryption and MACs
  - Record numbers prevent replay attacks within a connection


Client
Server
ClientHello
ServerHello
Certificate
$\{g^a \bmod p\}_{K^{-1}_{server}}$
Or RSA exchange
$g^b \bmod p$
$\{M, MAC(I_B, M)\}_{C_B}$
$\{M, MAC(I_S, M)\}_{C_S}$

TLS: Summary

- Security properties
  - DHE TLS: Forward secrecy
  - RSA TLS: No forward secrecy
  - End-to-end security: Secure even if all intermediate parties are malicious
  - Not anonymous: Attackers can determine who you’re talking to
  - No availability: Connections can be dropped or censored
- Can be used by the application layer (e.g. HTTPS)
- Trusting certificate authorities can be hard

DNS

Domain Names

- Recall: Computers are addressed by IP address on the Internet
  - Example: **`74.125.25.99`**
  - Useful for machines: Can be used to route packets to the correct destination
  - Not useful for humans: Numbers are not meaningful to humans, hard to remember
- More useful to humans: Human-readable domain names
  - Example: **`www.google.com`**
  - Not useful for machines: Contains no relevant routing information
  - Useful for humans: Meaningful words and phrases, easy to remember
  - Note: Domain names are not URLs. Domain names are part of a URL:
    `https://www.google.com/index.html`

DNS: Definition

- **DNS** (**Domain Name System**): An Internet protocol for translating human-readable domain names to IP addresses
- Usage
    - You want to send a packet to a certain domain (e.g. you type a domain into your browser)
    - Your computer performs a **DNS lookup** to translate the domain name to an IP address
    - Your computer sends the packet to the corresponding IP address

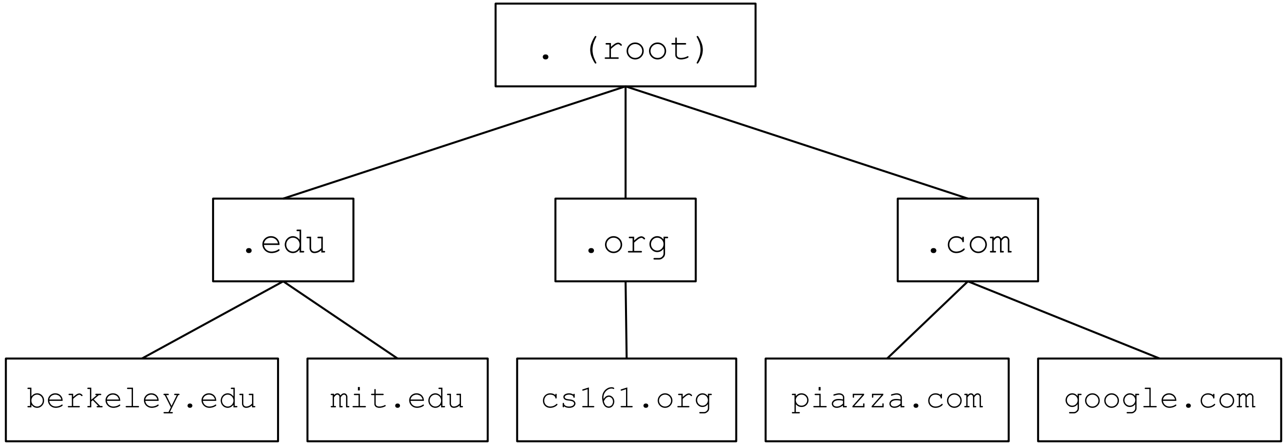
$$\texttt{www.google.com} \xrightarrow{\text{DNS}} \texttt{74.125.25.99}$$

DNS Name Servers

- **Name server**: A server on the Internet responsible for answering DNS requests
  - Name servers have domain names and IP addresses too
  - Example: Domain **`a.edu-servers.net`** with IP **`192.5.6.30`** is a name server
- Usage:
  - To perform a DNS lookup, your computer sends a **DNS query** (e.g. “What is the IP address of **`www.google.com`**?”)
  - The name server sends a **DNS response** with the answer (e.g. “The IP address of **`www.google.com`** is **`74.125.25.99`**”)
- Issues
  - One name server won’t be able to handle every DNS request from the entire Internet
  - If there are many name servers, how do you know which one to contact?

DNS Name Server Hierarchy

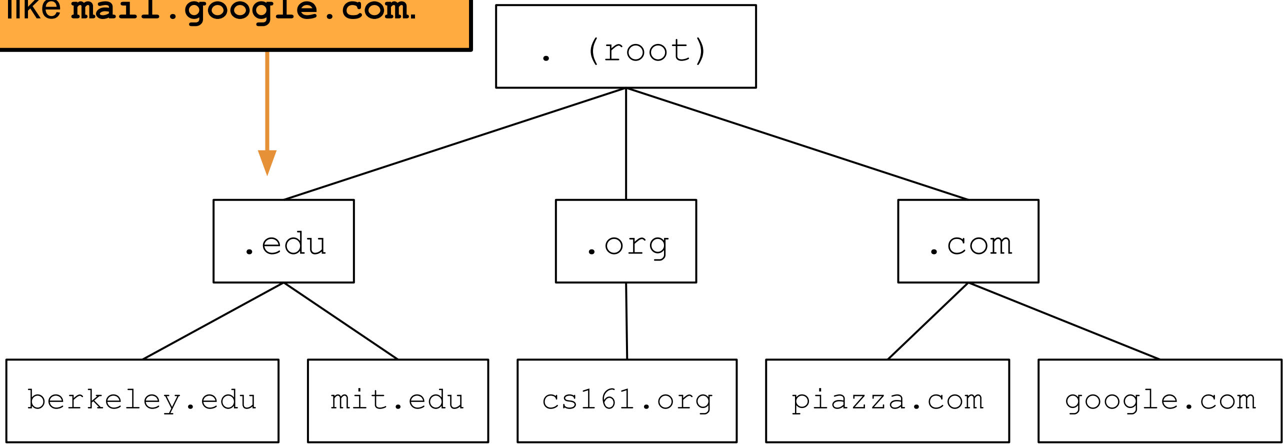
- Idea #1: If one name server doesn't know the answer to your query, the name server can direct you to another name server
    - Analogy: If I don't know the answer to your question, I will direct you to a friend who can help
- Idea #2: Arrange the name servers in a tree hierarchy
    - Intuition: Name servers will direct you down the tree until you receive the answer to your query



DNS Name Server Hierarchy

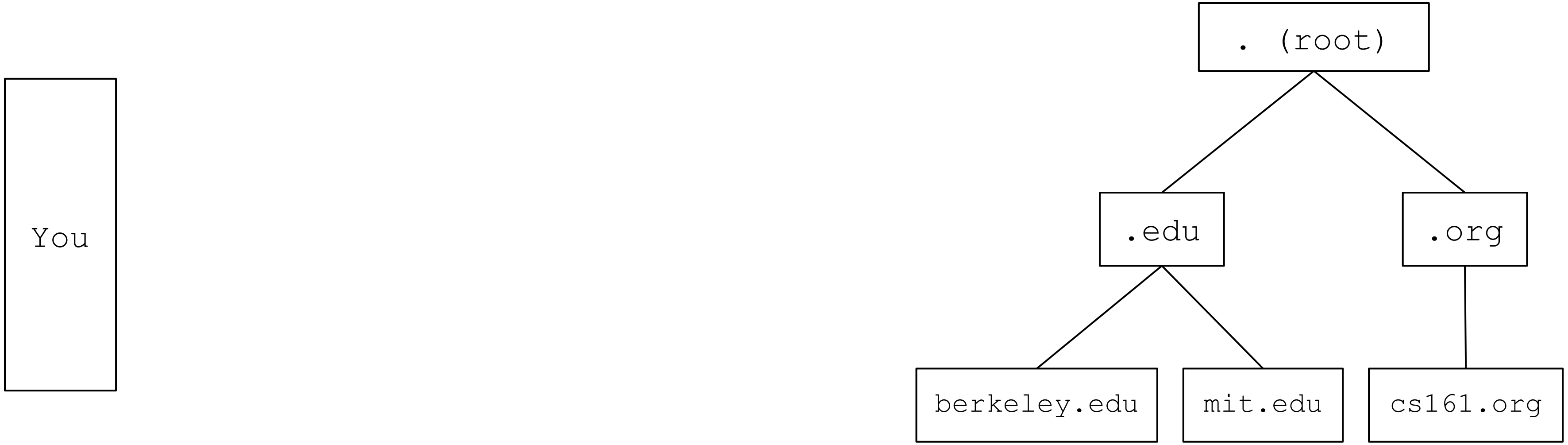
Each box is a name server. The label represents which queries the name server is responsible for answering.

For example, this name server is responsible for **`.edu`** queries like **`eecs.berkeley.edu`**, but not a query like **`mail.google.com`**.



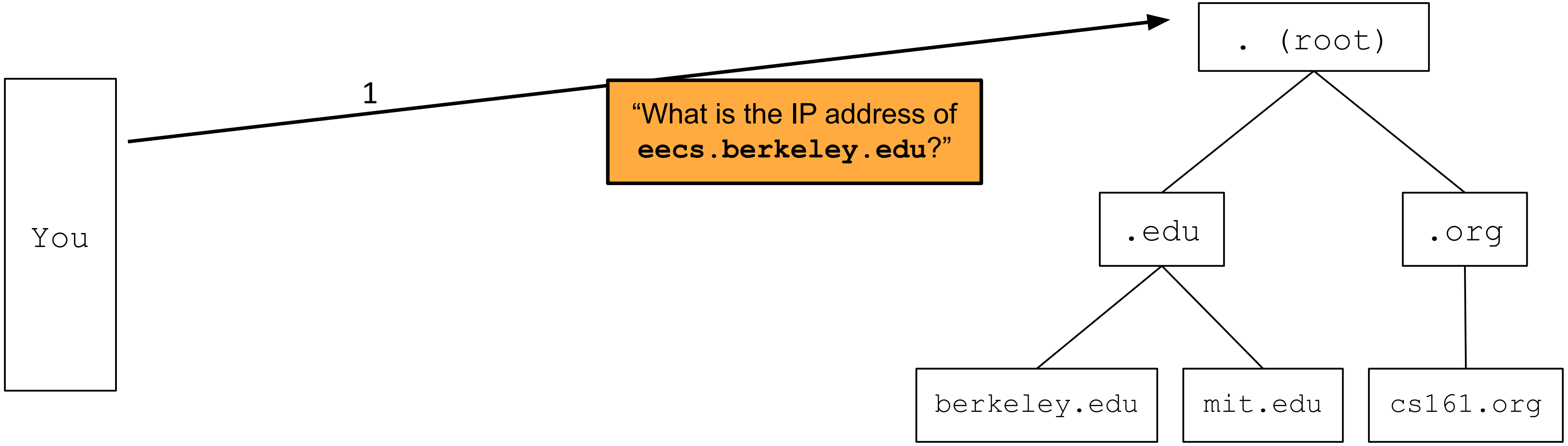
Steps of a DNS Lookup

Let's walk through a DNS query for the IP address of **`eecs.berkeley.edu`**.



Steps of a DNS Lookup

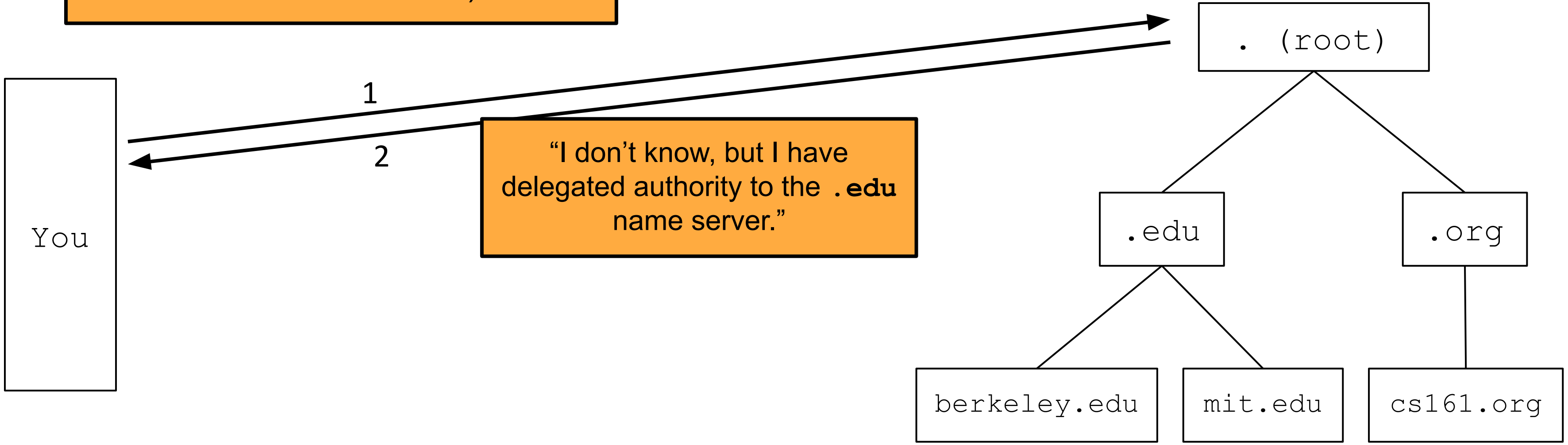
DNS queries always start with a request to the root name server, which is responsible for all requests.



"What is the IP address of **`eecs.berkeley.edu`**?"

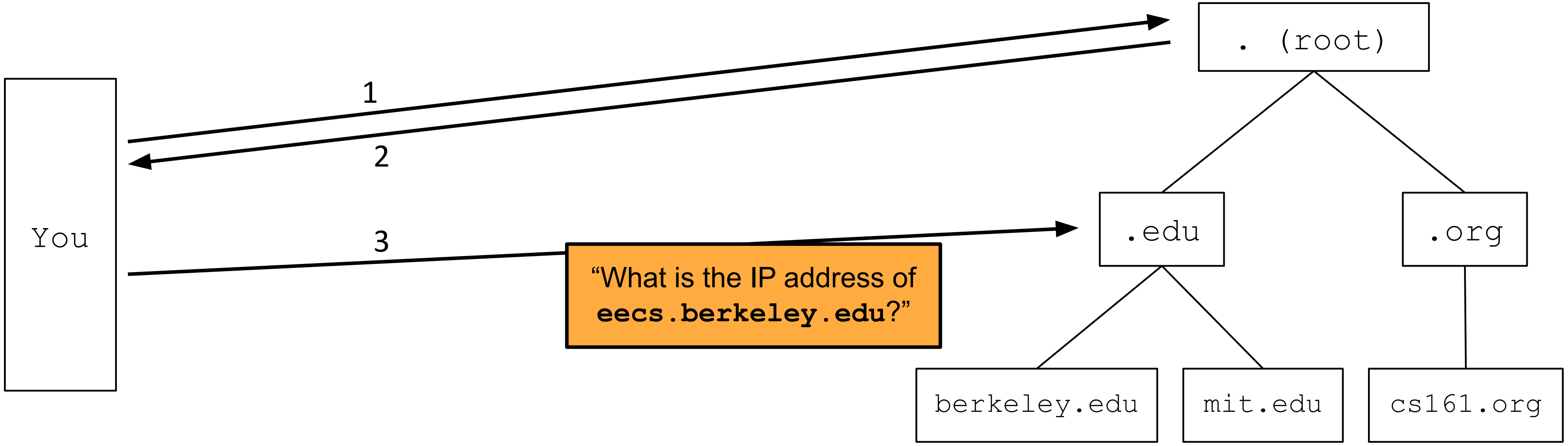
Steps of a DNS Lookup

The root name server responds by directing you to the correct child name server (in this case, the **`.edu`** name server).

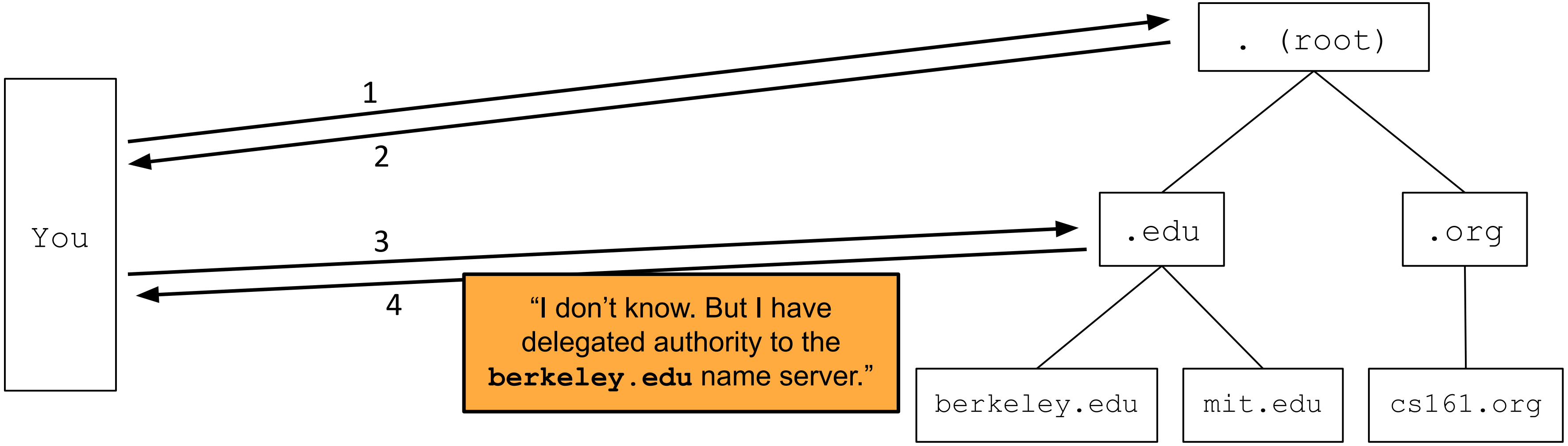


"I don't know, but I have delegated authority to the **`.edu`** name server."

Steps of a DNS Lookup

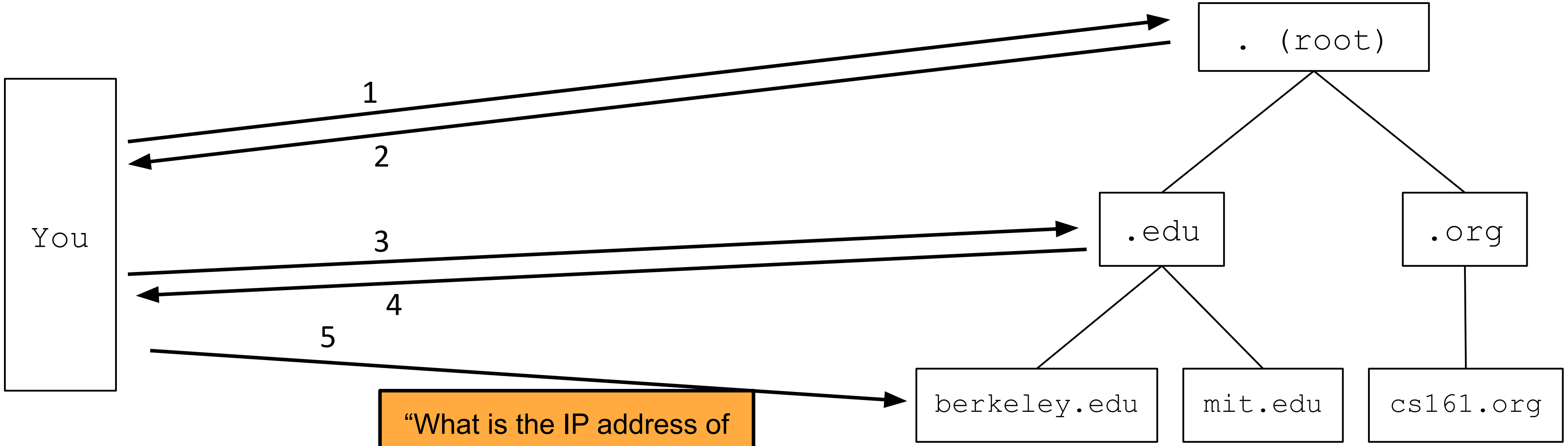


Steps of a DNS Lookup



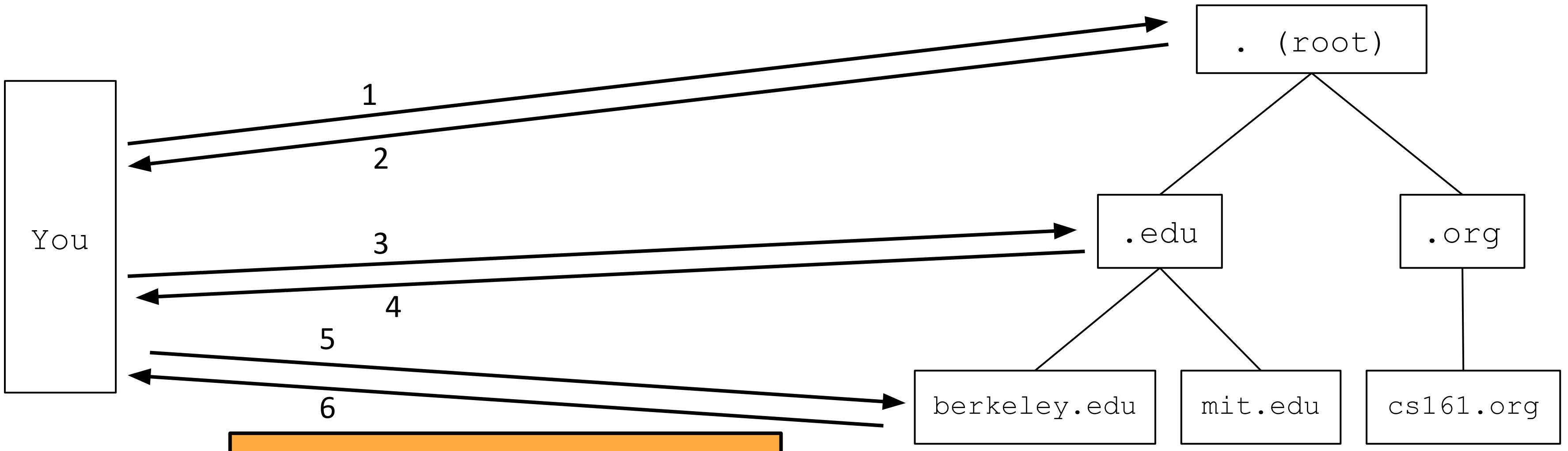
"I don't know. But I have delegated authority to the **berkeley.edu** name server."

Steps of a DNS Lookup



“What is the IP address of **`eecs.berkeley.edu`**?”

Steps of a DNS Lookup



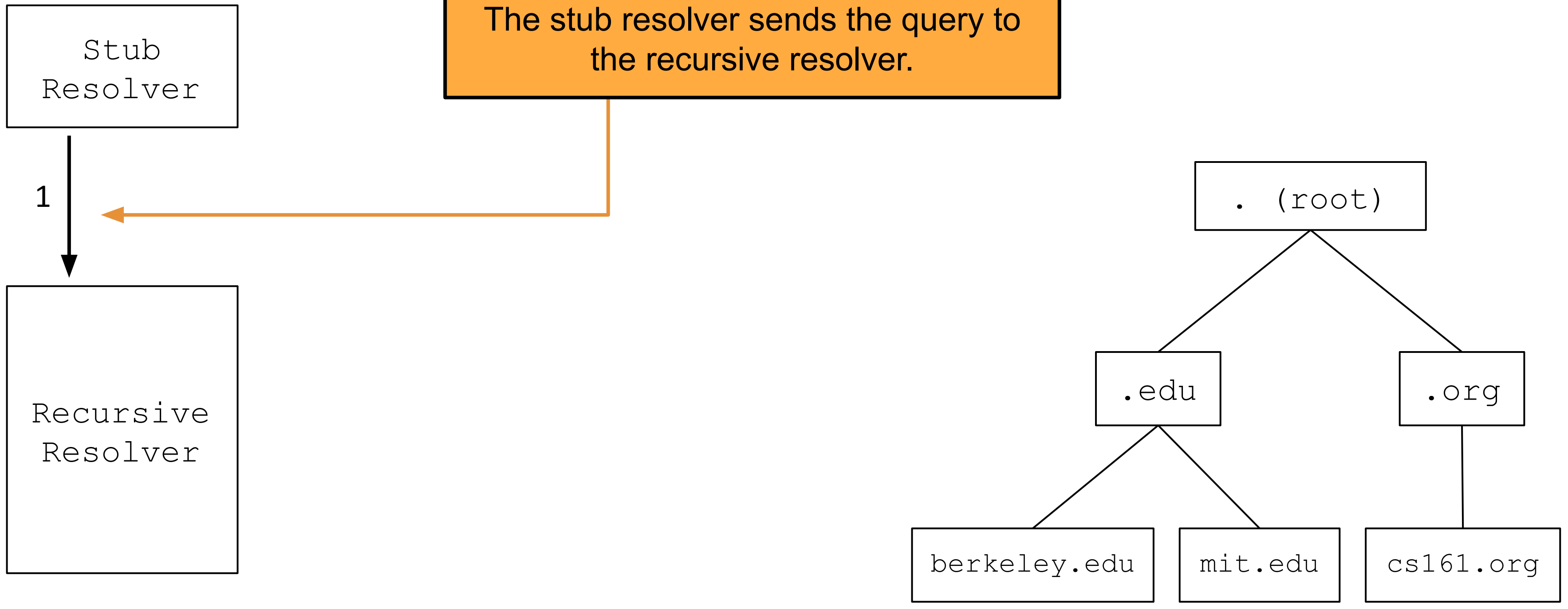
"The IP address of **eecs.berkeley.edu** is **23.185.0.1**."

Stub Resolvers and Recursive Resolvers

- In practice, your computer usually tells another resolver to perform the query for you
- **Stub resolver**: The resolver on your computer
  - Only contacts the recursive resolver and receives the answer
- **Recursive resolver**: The resolver that makes the actual DNS queries
  - Usually one recursive resolver per local network
  - Benefits: The recursive resolver can cache common requests for the network

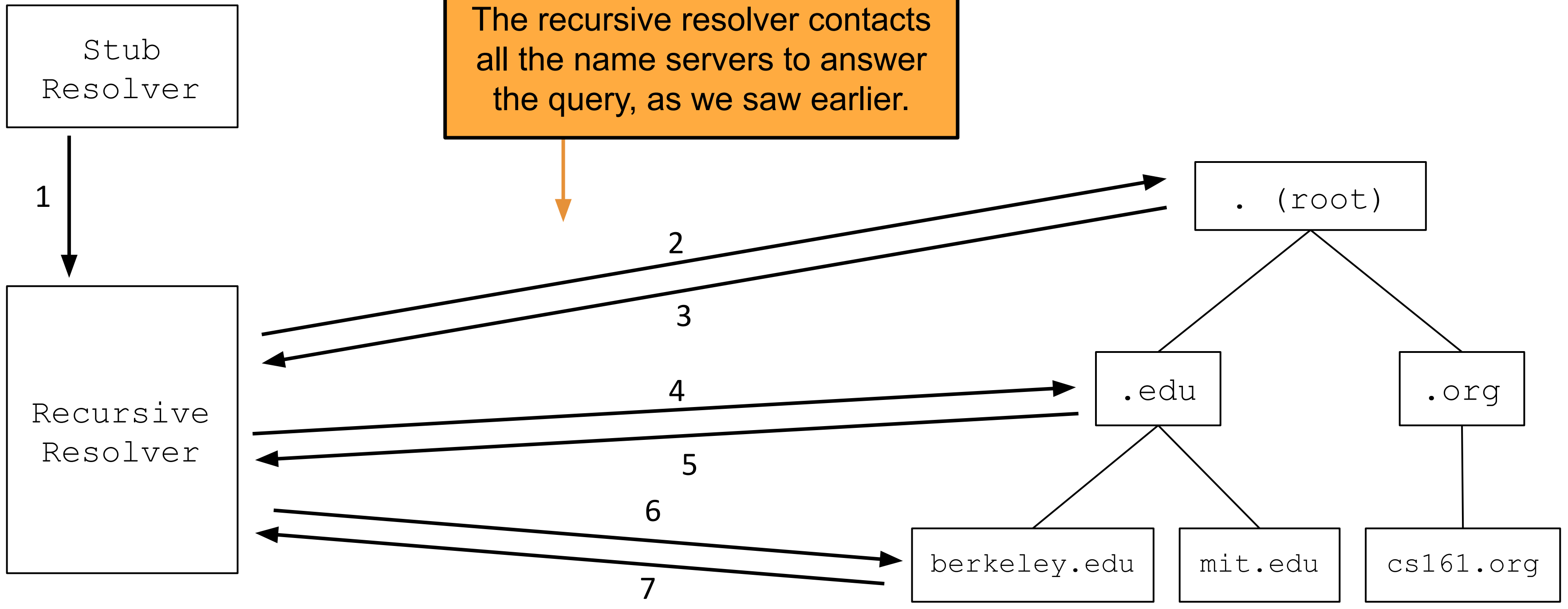
Steps of a DNS Lookup

The stub resolver sends the query to the recursive resolver.



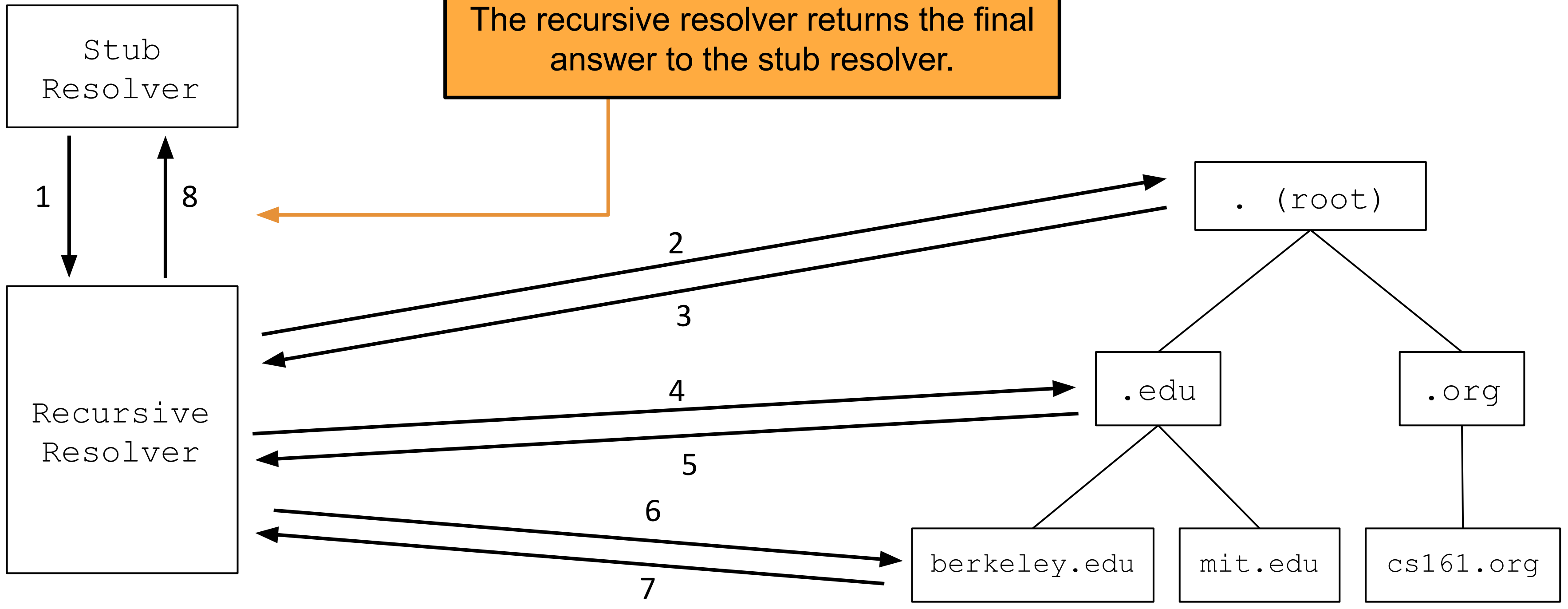
Steps of a DNS Lookup

The recursive resolver contacts all the name servers to answer the query, as we saw earlier.



Steps of a DNS Lookup

The recursive resolver returns the final answer to the stub resolver.



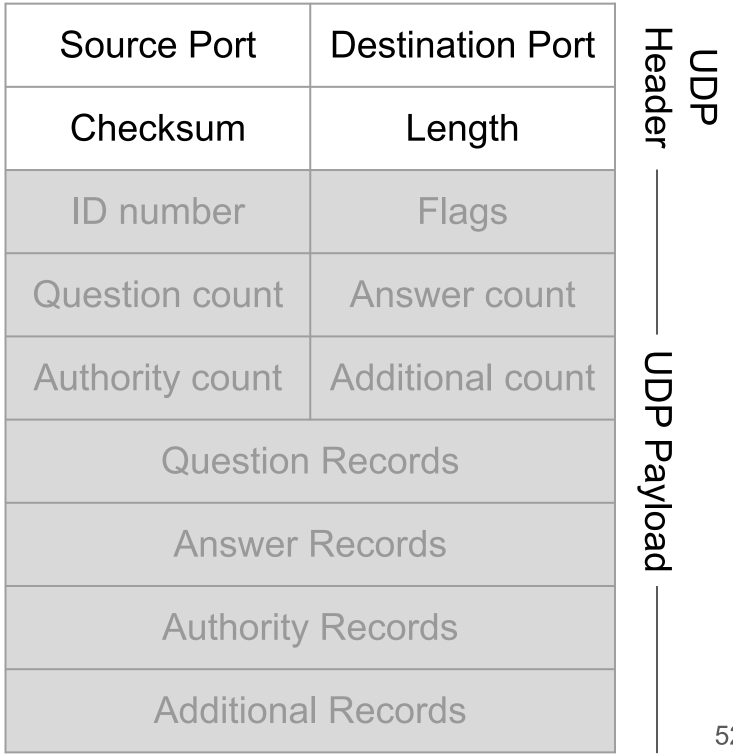
DNS Message Format

DNS Uses UDP

- Recall UDP vs. TCP
    - UDP: No delivery guarantees, packets can be reordered or dropped, faster
    - TCP: Packets guaranteed to arrive in order, slower
- DNS is designed to be lightweight and fast
    - Any access that involves a domain name (websites, email, etc.) is preceded by a DNS query, so we want DNS lookups to be fast
- DNS uses UDP instead of TCP for better performance
    - No 3-way handshake!

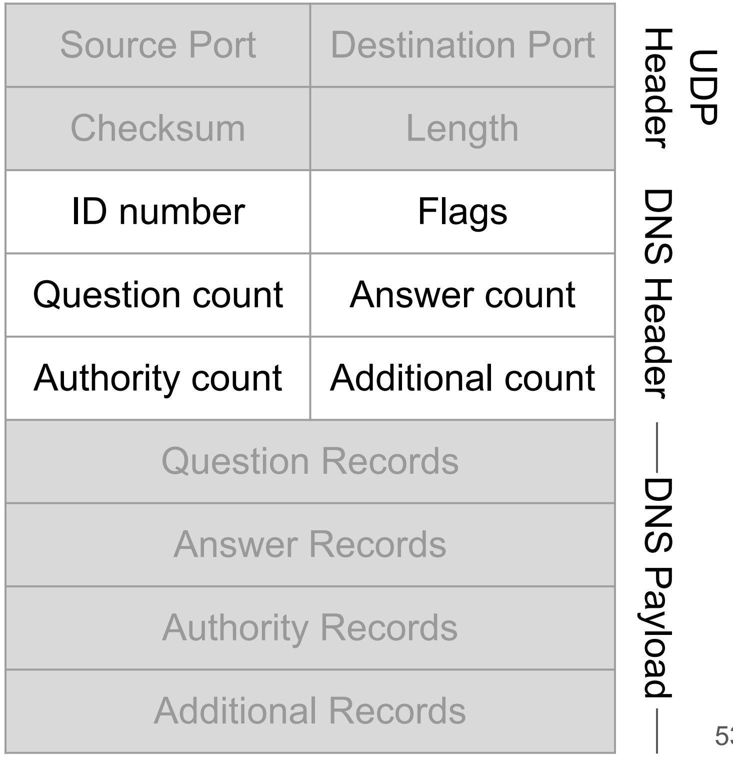
DNS Packet Format: UDP Header

- **Source port** (16 bits): Chosen by the client
  - Can be randomized for security, as we'll see later
- **Destination port** (16 bits): Usually 53
  - DNS name servers answer requests on Port 53
- Checksum: Code to check the UDP payload was not corrupted in transit
  - You don't need to worry about this
- Length: Length of the UDP payload
  - You don't need to worry about this



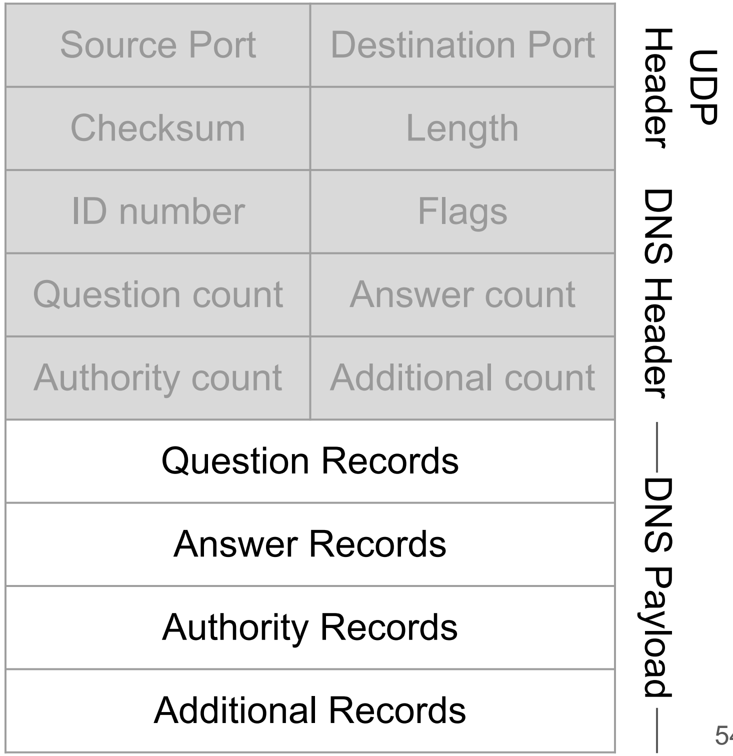
DNS Packet Format: DNS Payload

- **ID number** (16 bits): Used to associate queries with responses
    - Client picks an ID number in the query
    - Name server uses the same ID number in the response
    - Should be random for security, as we'll see later
- **Counts**: The number of records of each type in the DNS payload



DNS Packet Format: DNS Header

- The DNS payload contains a variable number of **resource records** (**RRs**)
- Each RR is a name-value pair
- RRs are sorted into four sections
  - Question section
  - Answer section
  - Authority section
  - Additional section



DNS Record Format

- Each record is a name-value pair with a type
    - **A (answer) type records**: Maps a domain name to an IPv4 address
    - **NS (name server) type records**: Designates another DNS server to handle a domain
    - Other types exist, but these are the two you need to know for now
- Each record also contains some metadata
    - **Time to live** (**TTL**): How long the record can be cached
    - Other metadata fields exist, but you don’t need to worry about them
    - All records of {name, type} pair form an **RRSET**, a group of records that should have the same TTL

DNS Record Types

- Other record types you might encounter:
  - **AAAA** type record: Maps a domain name to an IPv6 address
  - **CNAME** type record: Maps one domain name to another domain name. Used for aliases.
  - **MX** type record: Used for mail servers
  - **SOA**: Contains information about the operator/administrator of a zone
  - Other types for text records, cryptographic information, etc. exist too
  - You don’t need to know about any of these

DNS Record Sections

- Question section: What is being asked
  - Included in both requests and responses
  - Usually an A type record with the domain being looked up
- Answer section: A **direct response** to the question
  - Empty in requests
  - Used if the name server responds with the answer
  - Usually an A type record with the IP address of the domain being looked up
- Authority section: A **delegation of authority** for the question
  - Empty in requests
  - Used to direct the resolver to the next name server
  - Usually an NS type record with the zone and **domain** of the child name server
  - Additional:

DNS Record Sections

- Additional section: Additional information to help with the response, sometimes called **glue records**
  - Empty in requests
  - Provides helpful, **non-authoritative** records for domains
  - Usually an A type record with the domain and IP address of the child name server (since the NS record provides the child name server as a **domain**)

DNS Record Caching

- For performance, resolvers cache as many records as possible
  - Records returned by name servers are cached until their time-to-live expires
  - No DNS requests need to be sent for recently-seen queries
  - Makes response time faster for clients
  - Reduces load on name servers

DNS Lookup Walkthrough

```
$ dig +norecurse eecs.berkeley.edu @198.41.0.4
```

You can try this at home! Use the **`dig`** utility in your terminal, and remember to set the **`+norecurse`** flag so you can traverse the name server hierarchy yourself.

DNS Lookup Walkthrough

```
$ dig +norecurse eecs.berkeley.edu @198.41.0.4
```

We are performing a DNS lookup for the IP address of **`eecs.berkeley.edu`**.

DNS Lookup Walkthrough

```
$ dig +norecurse eecs.berkeley.edu @198.41.0.4
```

DNS queries always start with a request to the root name server. The IP address of the root name server is usually hard-coded into recursive resolvers.

DNS Lookup Walkthrough

```
$ dig +norecurse eecs.berkeley.edu @198.41.0.4
```

```
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 26114
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 13, ADDITIONAL: 27

;; QUESTION SECTION:
;eecs.berkeley.edu.             IN    A

;; AUTHORITY SECTION:
edu.                    172800  IN    NS    a.edu-servers.net.
edu.                    172800  IN    NS    b.edu-servers.net.
edu.                    172800  IN    NS    c.edu-servers.net.
...

;; ADDITIONAL SECTION:
a.edu-servers.net.      172800  IN    A      192.5.6.30
b.edu-servers.net.      172800  IN    A      192.33.14.30
c.edu-servers.net.      172800  IN    A      192.26.92.30
...
```

Here's the DNS response from the root name server.

DNS Lookup Walkthrough

```
$ dig +norecurse eecs.berkeley.edu @198.41.0.4

;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 26114
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 13, ADDITIONAL: 27

;; QUESTION SECTION:
;eecs.berkeley.edu.             IN    A

;; AUTHORITY SECTION:
edu.                    172800  IN    NS    a.edu-servers.net.
edu.                    172800  IN    NS    b.edu-servers.net.
edu.                    172800  IN    NS    c.edu-servers.net.
...

;; ADDITIONAL SECTION:
a.edu-servers.net.      172800  IN    A      192.5.6.30
b.edu-servers.net.      172800  IN    A      192.33.14.30
c.edu-servers.net.      172800  IN    A      192.26.92.30
...
```

Here's the DNS header.

DNS Lookup Walkthrough

```
$ dig +norecurse eecs.berkeley.edu @198.41.0.4

;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 26114
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 13, ADDITIONAL: 27

;; QUESTION SECTION:
;eecs.berkeley.edu.            IN    A

;; AUTHORITY SECTION:
edu.                   172800  IN    NS    a.edu-servers.net.
edu.                   172800  IN    NS    b.edu-servers.net.
edu.                   172800  IN    NS    c.edu-servers.net.
...

;; ADDITIONAL SECTION:
a.edu-servers.net.     172800  IN    A      192.5.6.30
b.edu-servers.net.     172800  IN    A      192.33.14.30
c.edu-servers.net.     172800  IN    A      192.26.92.30
...
```

Here's the 16-bit ID number in the DNS header.

DNS Lookup Walkthrough

```
$ dig +norecurse eecs.berkeley.edu @198.41.0.4

;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 26114
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 13, ADDITIONAL: 27

;; QUESTION SECTION:
;eecs.berkeley.edu.            IN    A

;; AUTHORITY SECTION:
edu.                   172800  IN    NS    a.edu-servers.net.
edu.                   172800  IN    NS    b.edu-servers.net.
edu.                   172800  IN    NS    c.edu-servers.net.
...

;; ADDITIONAL SECTION:
a.edu-servers.net.     172800  IN    A      192.5.6.30
b.edu-servers.net.     172800  IN    A      192.33.14.30
c.edu-servers.net.     172800  IN    A      192.26.92.30
...
```

Here are the record counts in the DNS header.

Here are the flags in the DNS header.

DNS Lookup Walkthrough

```
$ dig +norecurse eecs.berkeley.edu @198.41.0.4

;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 26114
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 13, ADDITIONAL: 27
;; QUESTION SECTION:
;eecs.berkeley.edu.          IN    A

;; AUTHORITY SECTION:
edu.                 172800  IN    NS    a.edu-servers.net.
edu.                 172800  IN    NS    b.edu-servers.net.
edu.                 172800  IN    NS    c.edu-servers.net.
...

;; ADDITIONAL SECTION:
a.edu-servers.net.   172800  IN    A      192.5.6.30
b.edu-servers.net.   172800  IN    A      192.33.14.30
c.edu-servers.net.   172800  IN    A      192.26.92.30
...
```

Here's the DNS payload. It's a collection of resource records (one per line), sorted into four sections.

DNS Lookup Walkthrough

```
$ dig +norecurse eecs.berkeley.edu @198.41.0.4

;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 26114
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 13, ADDITIONAL: 27

;; QUESTION SECTION:
;eecs.berkeley.edu.             IN    A

;; AUTHORITY SECTION:
edu.                    172800  IN    NS    a.edu-servers.net.
edu.                    172800  IN    NS    b.edu-servers.net.
edu.                    172800  IN    NS    c.edu-servers.net.
...

;; ADDITIONAL SECTION:
a.edu-servers.net.      172800  IN    A      192.5.6.30
b.edu-servers.net.      172800  IN    A      192.33.14.30
c.edu-servers.net.      172800  IN    A      192.26.92.30
...
```

Here's the question section. The name is **`eecs.berkeley.edu`**, the type is A, and the value is blank. It shows that we are looking for the IP address of **`eecs.berkeley.edu`**.

DNS Lookup Walkthrough

```
$ dig +norecurse eecs.berkeley.edu @198.41.0.4

;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 26114
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 13, ADDITIONAL: 27

;; QUESTION SECTION:
;eecs.berkeley.edu.            IN    A

;; AUTHORITY SECTION:
edu.                    172800 IN    NS    a.edu-servers.net.
edu.                    172800 IN    NS    b.edu-servers.net.
edu.                    172800 IN    NS    c.edu-servers.net.
...

;; ADDITIONAL SECTION:
a.edu-servers.net.      172800 IN    A      192.5.6.30
b.edu-servers.net.      172800 IN    A      192.33.14.30
c.edu-servers.net.      172800 IN    A      192.26.92.30
...
```

The answer section is blank, because the root name server did not return the answer we're looking for.

We can confirm this by checking the header, which says there are 0 records in the answer section.

DNS Lookup Walkthrough

```
$ dig +norecurse eecs.berkeley.edu @198.41.0.4

;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 26114
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 13, ADDITIONAL: 27

;; QUESTION SECTION:
;eecs.berkeley.edu.              IN    A

;; AUTHORITY SECTION:
edu.                    172800   IN    NS    a.edu-servers.net.
edu.                    172800   IN    NS    b.edu-servers.net.
edu.                    172800   IN    NS    c.edu-servers.net.
...

;; ADDITIONAL SECTION:
a.edu-servers.net.      172800   IN    A      192.5.6.30
b.edu-servers.net.      172800   IN    A      192.33.14.30
c.edu-servers.net.      172800   IN    A      192.26.92.30
...
```

The authority and additional sections tell the resolver where to look next.

Note that there are multiple **`.edu`** name servers for redundancy.

DNS Lookup Walkthrough

```
$ dig +norecurse eecs.berkeley.edu @198.41.0.4

;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 26114
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 13, ADDITIONAL: 27

;; QUESTION SECTION:
;eecs.berkeley.edu.              IN    A

;; AUTHORITY SECTION:
edu.                    172800   IN    NS    a.edu-servers.net.
edu.                    172800   IN    NS    b.edu-servers.net.
edu.                    172800   IN    NS    c.edu-servers.net.
...

;; ADDITIONAL SECTION:
a.edu-servers.net.  172800   IN    A      192.5.6.30
b.edu-servers.net.  172800   IN    A      192.33.14.30
c.edu-servers.net.  172800   IN    A      192.26.92.30
...
```

For redundancy, there are usually several name servers for each zone. Any of them will usually work. Let's pick the first one.

This NS record says that **`a.edu-servers.net`** is a **`.edu`** name server.

DNS Lookup Walkthrough

```
$ dig +norecurse eecs.berkeley.edu @198.41.0.4

;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 26114
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 13, ADDITIONAL: 27

;; QUESTION SECTION:
;eecs.berkeley.edu.          IN    A

;; AUTHORITY SECTION:
edu.                172800   IN    NS    a.edu-servers.net.
edu.                172800   IN    NS    b.edu-servers.net.
edu.                172800   IN    NS    c.edu-servers.net.
...

;; ADDITIONAL SECTION:
a.edu-servers.net.  172800   IN    A       192.5.6.30
b.edu-servers.net.  172800   IN    A       192.33.14.30
c.edu-servers.net.  172800   IN    A       192.26.92.30
...
```

This A record helpfully tells us the IP address of the next name server we mean to contact.

DNS Lookup Walkthrough

```
$ dig +norecurse eecs.berkeley.edu @192.5.6.30
```

Next, we query the **`.edu`** name server. We know the IP address of the **`.edu`** name server because the root name server gave the information to us.

DNS Lookup Walkthrough

```
$ dig +norecurse eecs.berkeley.edu @192.5.6.30

;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 36257
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 3, ADDITIONAL: 5

;; QUESTION SECTION:
;eecs.berkeley.edu.            IN    A

;; AUTHORITY SECTION:
berkeley.edu.        172800    IN    NS    adns1.berkeley.edu.
berkeley.edu.        172800    IN    NS    adns2.berkeley.edu.
berkeley.edu.        172800    IN    NS    adns3.berkeley.edu.

;; ADDITIONAL SECTION:
adns1.berkeley.edu.  172800    IN    A     128.32.136.3
adns2.berkeley.edu.  172800    IN    A     128.32.136.14
adns3.berkeley.edu.  172800    IN    A     192.107.102.142
...
```

The answer section is blank again. The authority and additional section tell us to query a **`berkeley.edu`** name server, and provide us with the IP address of the next name server.

DNS Lookup Walkthrough

```
$ dig +norecurse eecs.berkeley.edu @128.32.136.3
```

Next, we query the **`berkeley.edu`** name server for the IP address of **`eecs.berkeley.edu`**. We know the IP address of the **`berkeley.edu`** name server because the root name server gave the information to us.

DNS Lookup Walkthrough

```
$ dig +norecurse eecs.berkeley.edu @128.32.136.3

;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 52788
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; QUESTION SECTION:
;eecs.berkeley.edu.          IN   A

;; ANSWER SECTION:
eecs.berkeley.edu.  86400    IN   A    23.185.0.1
```

The answer section has one A type record. It tells us that the IP address of **`eecs.berkeley.edu`** is **`23.185.0.1`**.

DNS Lookup Walkthrough

```
$ dig +norecurse eecs.berkeley.edu @128.32.136.3

;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 52788
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; QUESTION SECTION:
;eecs.berkeley.edu.            IN   A

;; ANSWER SECTION:
eecs.berkeley.edu.  86400      IN   A   23.185.0.1
```

Here's the time-to-live (TTL) field in the record. It tells us that we can cache this answer for 86,400 seconds (24 hours).

DNS Security

Cache Poisoning Attacks

- **Cache poisoning attack**: Returning a malicious record to the client
  - The victim will cache the malicious records, “poisoning” it
- Example: Supply a malicious A record mapping the attacker’s IP address to a legitimate domain
  - Now when the victim visits **`eecs.berkeley.edu`**, they’ll actually be sending packets to the attacker (**`6.6.6.6`**), who can act as a MITM!

Security Risk: Malicious Name Servers

- Malicious name servers can lie and supply a malicious answer
- Malicious records could also poison the cache with other records

```
$ dig +norecurse eecs.berkeley.edu @128.32.136.3

;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 52788
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; QUESTION SECTION:
;eecs.berkeley.edu.              IN    A

;; ANSWER SECTION:
eecs.berkeley.edu.    86400      IN    A     23.185.0.1

;; ADDITIONAL SECTION:
www.google.com.       172800     IN    A     6.6.6.6
```

We made a query to a malicious **berkeley.edu** name server...

...and it returned a malicious record for **www.google.com**!

Defense: Bailiwick Checking

- Idea: Limit the amount of damage a malicious name server can do
- **Bailiwick checking**: the resolver only accepts records if they are in the name server's zone
  - Bailiwick: "one's sphere of operations or particular area of interest"
  - Example: The **`berkeley.edu`** name server can provide a record for **`eecs.berkeley.edu`**, but not **`mit.edu`**
  - Example: The **`.edu`** name server can provide a record for **`mit.edu`** and **`berkeley.edu`**, but not **`google.com`**
  - Example: The root name server can provide a record for any domain (everything is in bailiwick for the root)

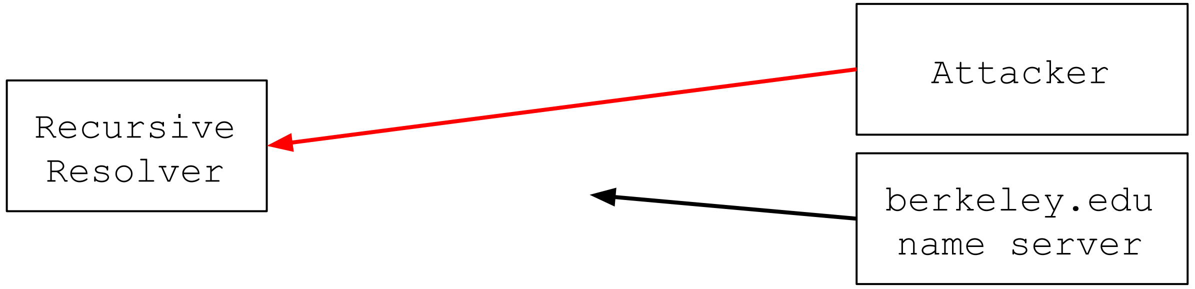
Security Risk: Man-in-the-middle (MITM) Attackers

- DNS is not secure against MITM attackers
- MITM attackers can poison the cache by adding, removing, or changing any record in the DNS response

```
;; ANSWER SECTION:
eecs.berkeley.edu.   86400   IN   A   ~~23.185.0.1~~ 6.6.6.6
```

Security Risk: On-Path Attackers

- DNS is not secure against on-path attackers
- On-path attackers can poison the cache by sending a spoofed response
  - If the spoofed response arrives before the legitimate response, the victim will cache the attacker's malicious records
  - The on-path attacker can see every field in the unencrypted DNS request. Nothing to guess!


Recursive Resolver
Attacker
berkeley.edu name server