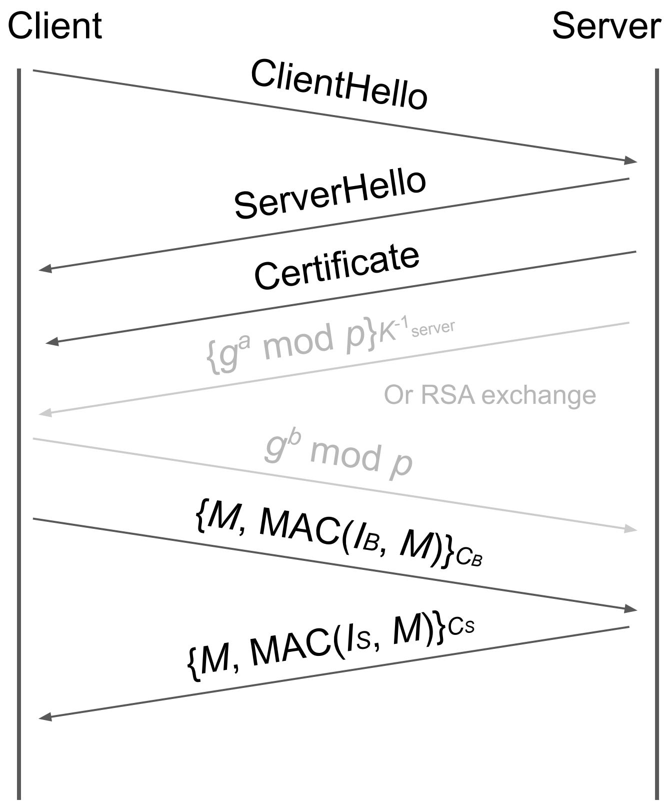
DNS

CS 161 Spring 2024 - Lecture 20

Last Time: TLS

- 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



Last Time: TLS

- 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

Outline

- Domain Name System (DNS)
  - DNS name servers
  - Steps of a DNS lookup
  - Stub resolvers and recursive resolvers
  - DNS message format
  - DNS records
  - DNS lookup walkthrough
- DNS Security
  - Cache poisoning attacks
  - Risk: Malicious name servers
  - Defense: Bailiwick checking
  - Risk: Network attackers (MITM, on-path, off-path)
  - Kaminsky attack
  - Defense: Source port randomization

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

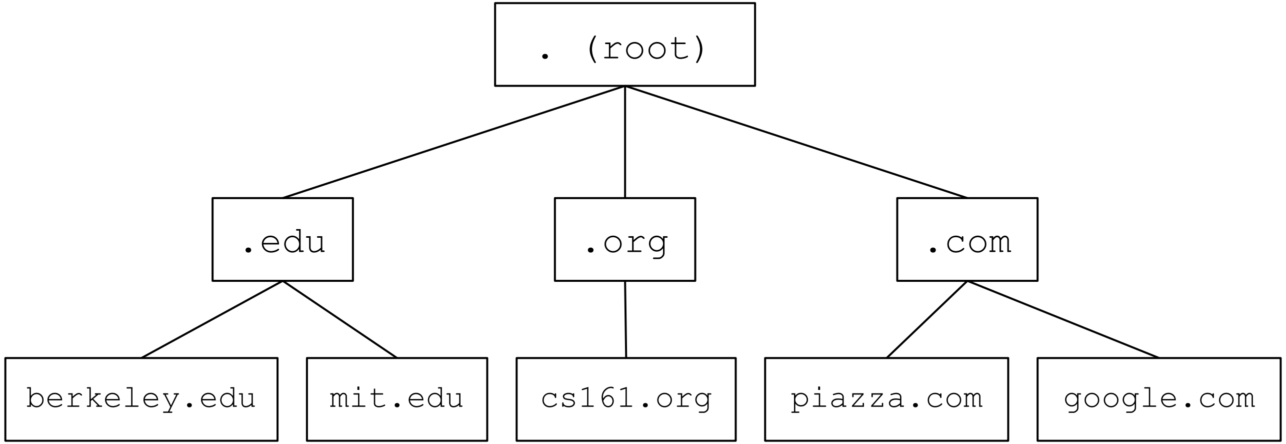
`www.google.com` —DNS→ **`74.125.25.99`**

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

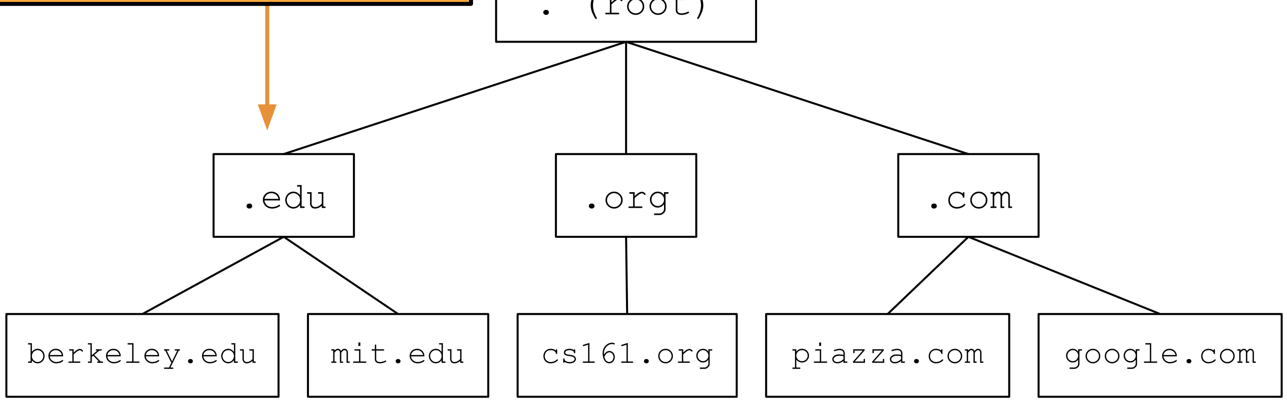
- Idea #1: If one name server doesn’t know the answer to your query, the name server can direct you to another name server
  - Analogy: If I don't know the answer to your question, I will direct you to a friend who can help
- Idea #2: Arrange the name servers in a tree hierarchy
  - Intuition: Name servers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



DNS Name Server Hierarchy

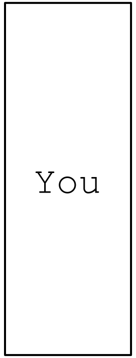
Each box is a name server. The label represents which queries the name server is responsible for answering.

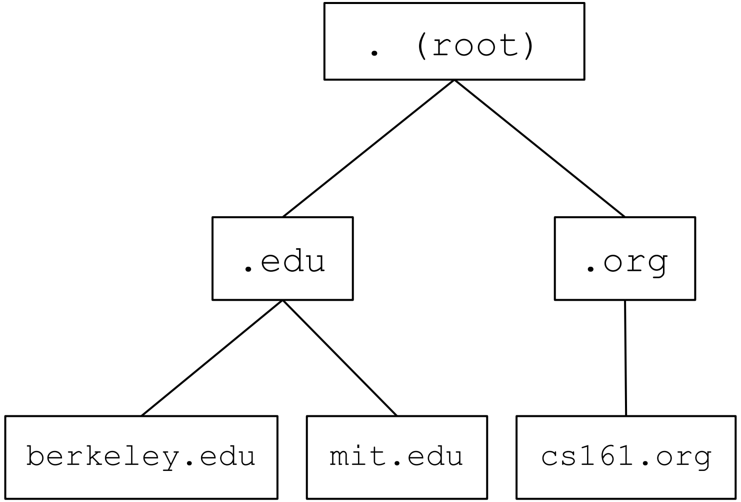
For example, this name server is responsible for **`.edu`** queries like **`eecs.berkeley.edu`**, but not a query like **`mail.google.com`**.



Steps of a DNS Lookup

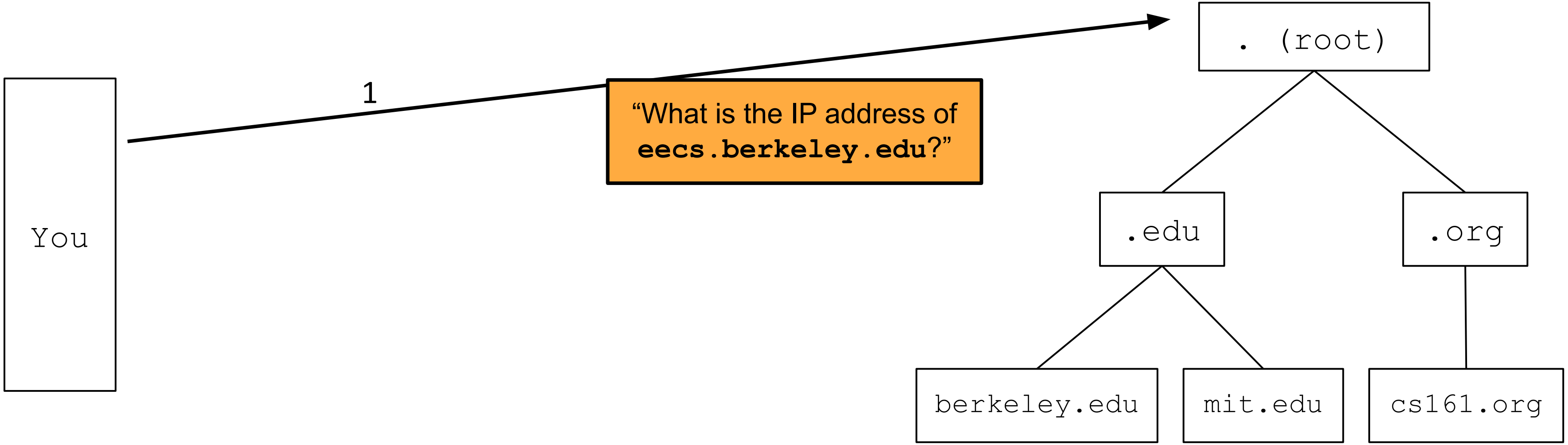
Let's walk through a DNS query for the IP address of **`eecs.berkeley.edu`**.





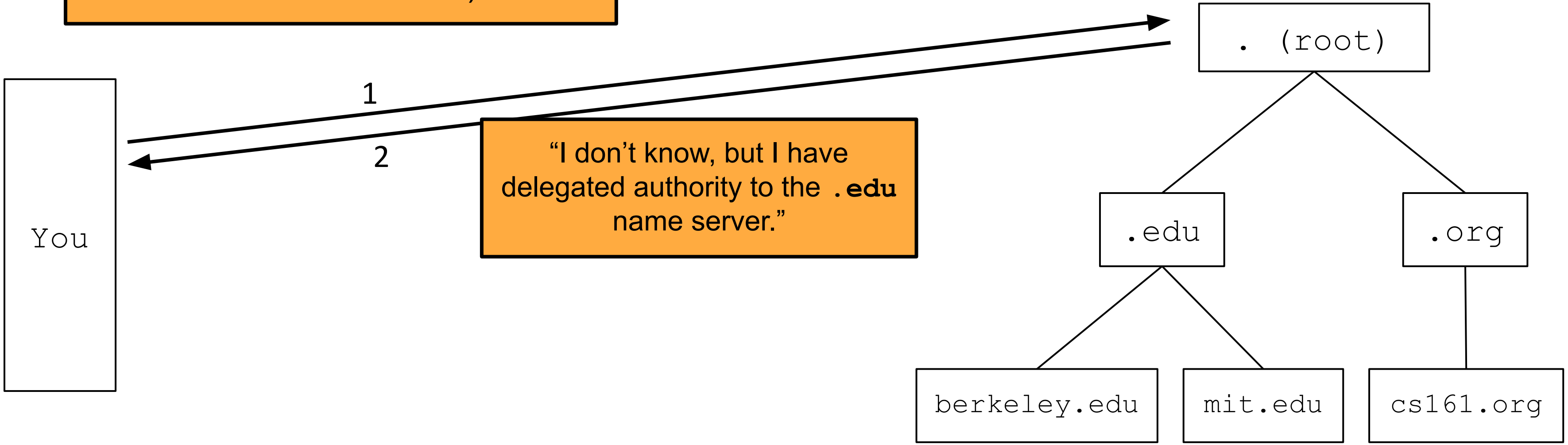
Steps of a DNS Lookup

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

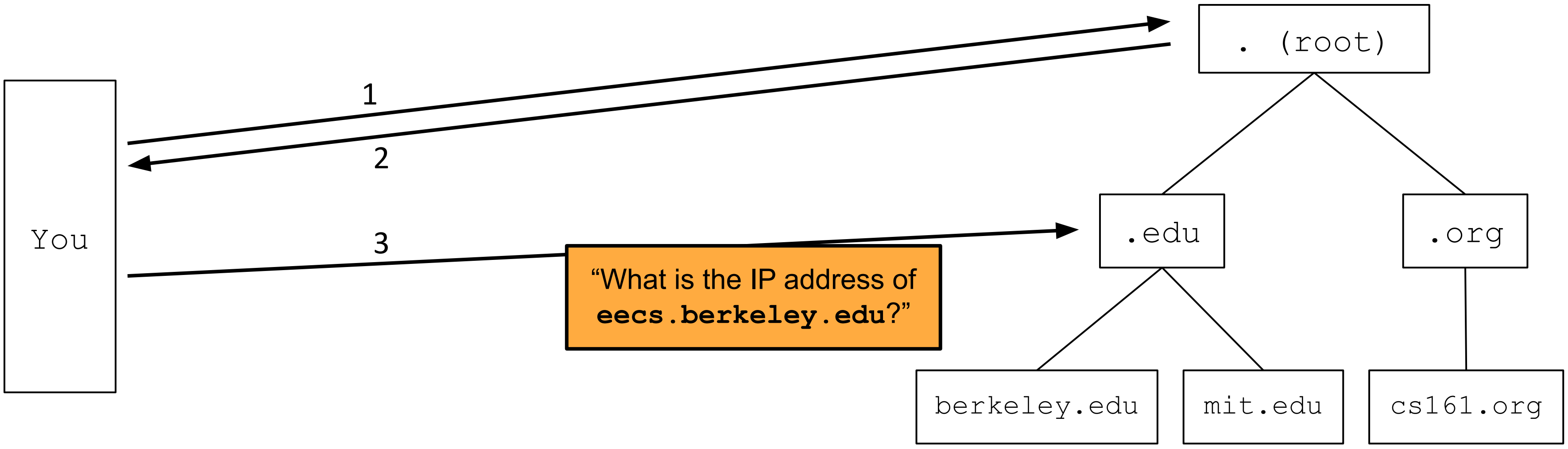


Steps of a DNS Lookup

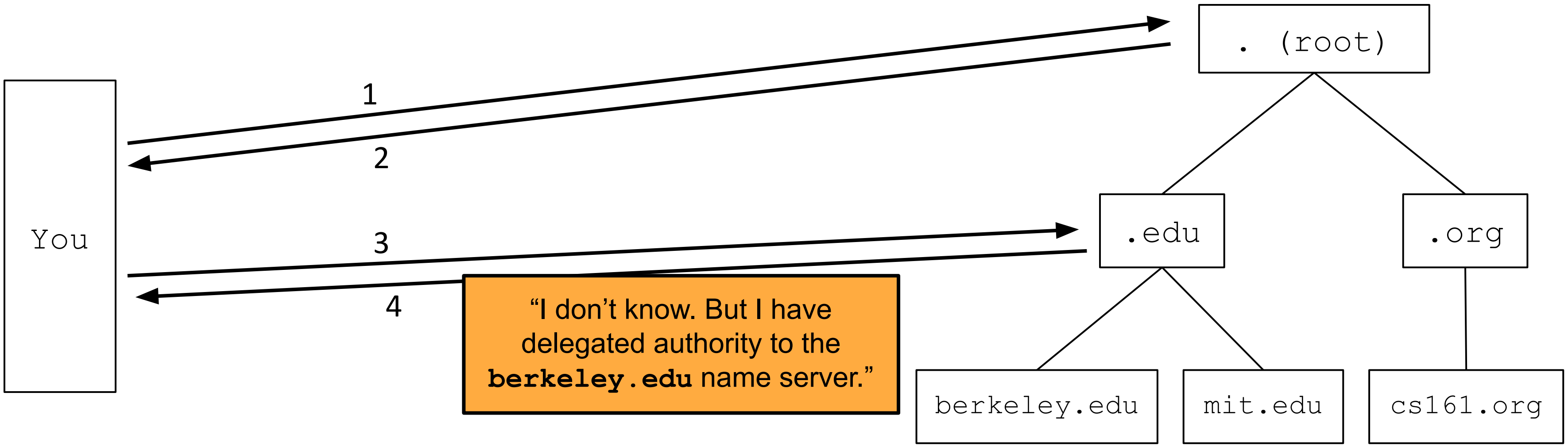
The root name server responds by directing you to the correct child name server (in this case, the **`.edu`** name server).



Steps of a DNS Lookup

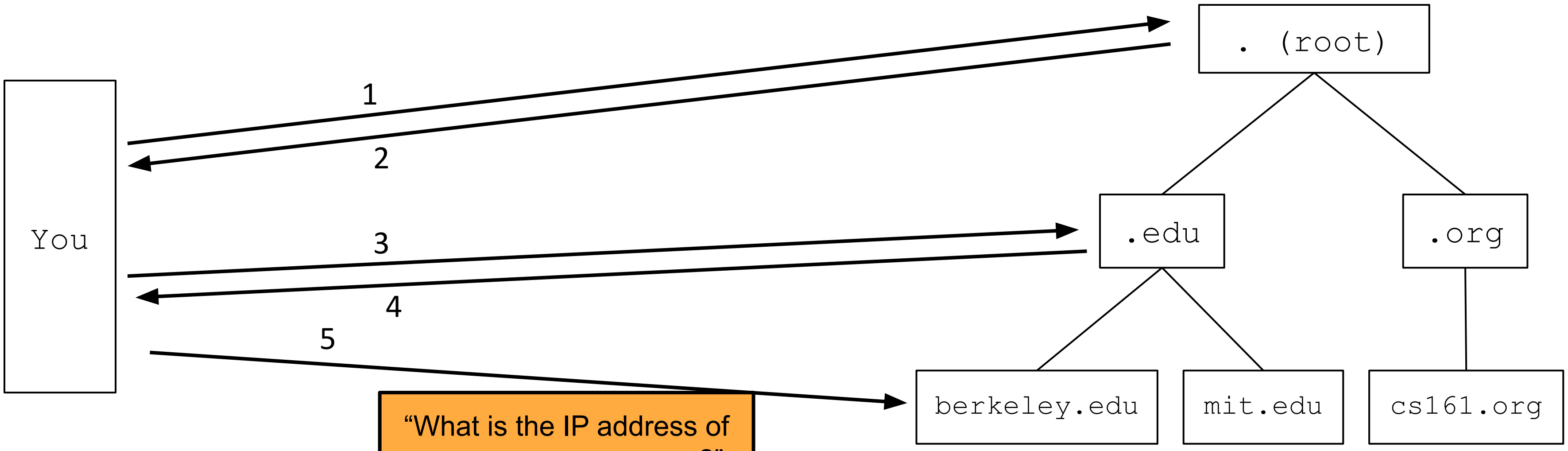


Steps of a DNS Lookup



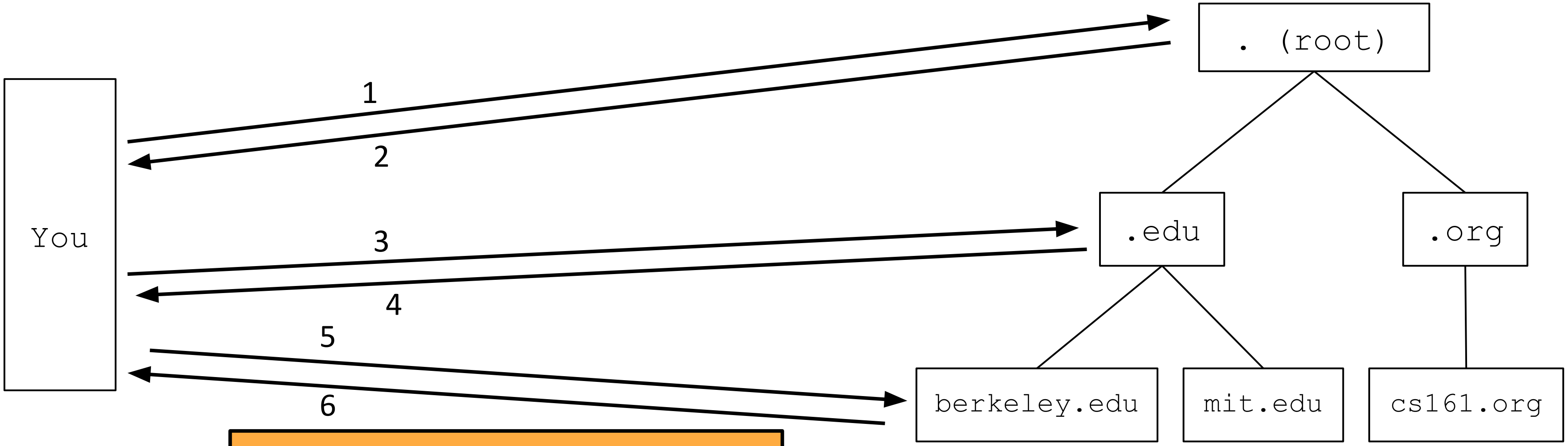
“I don’t know. But I have delegated authority to the **`berkeley.edu`** name server.”

Steps of a DNS Lookup



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

Steps of a DNS Lookup

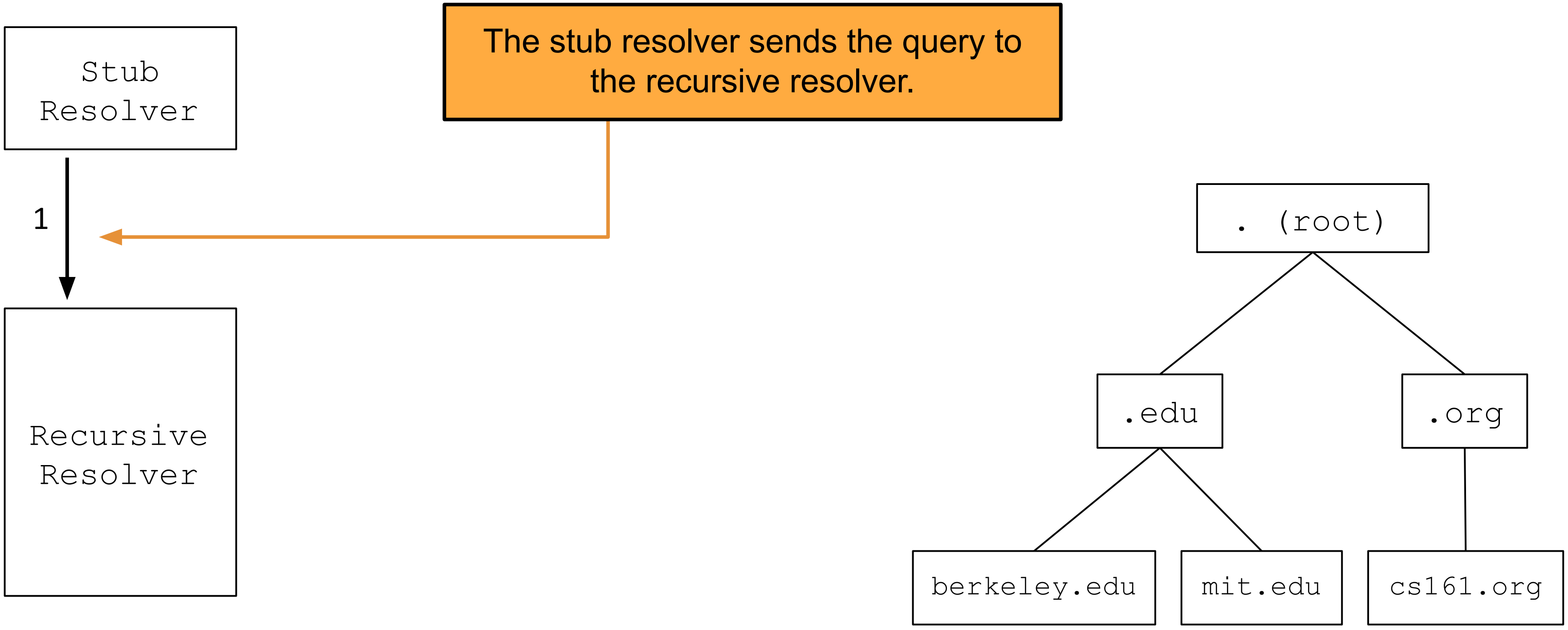


"The IP address of **`eecs.berkeley.edu`** is **`23.185.0.1`**."

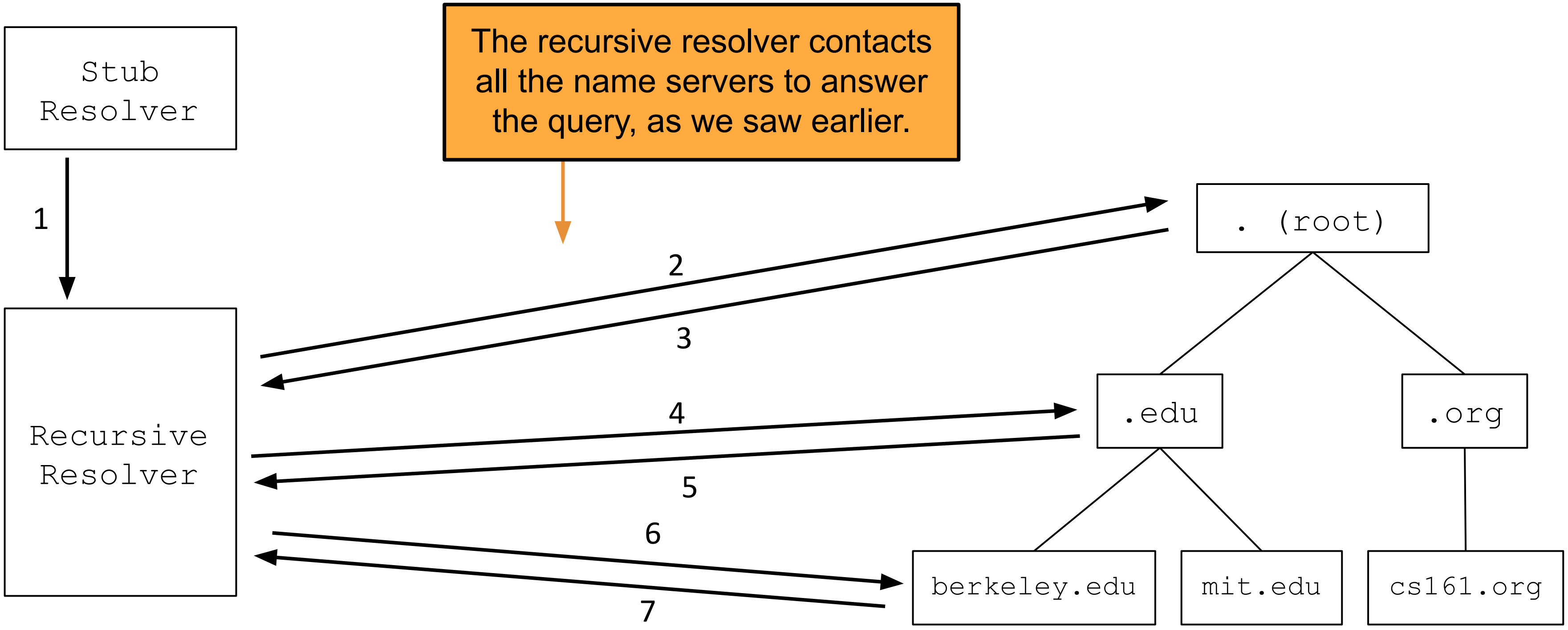
Stub Resolvers and Recursive Resolvers

- 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

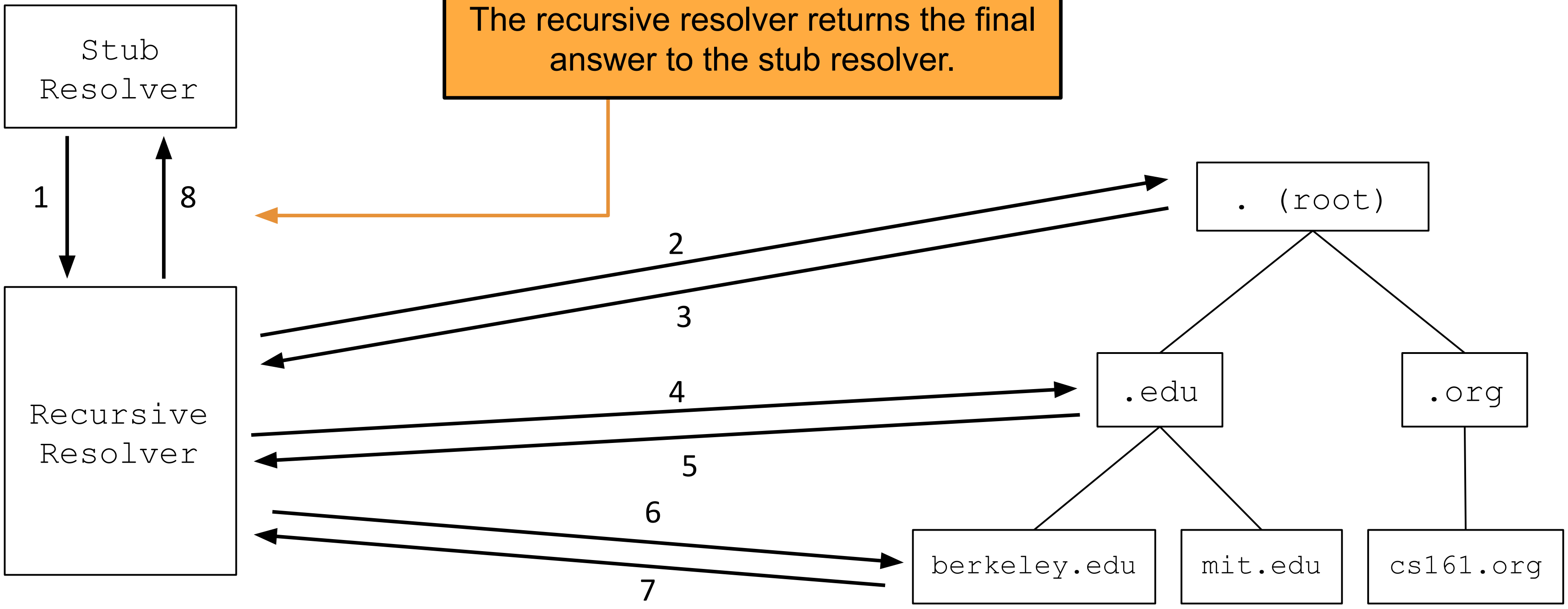


Steps of a DNS Lookup



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

Source Port	Destination Port	UDP Header
Checksum	Length	
ID number	Flags	UDP Payload
Question count	Answer count	
Authority count	Additional count	
Question Records		
Answer Records		
Authority Records		
Additional Records		

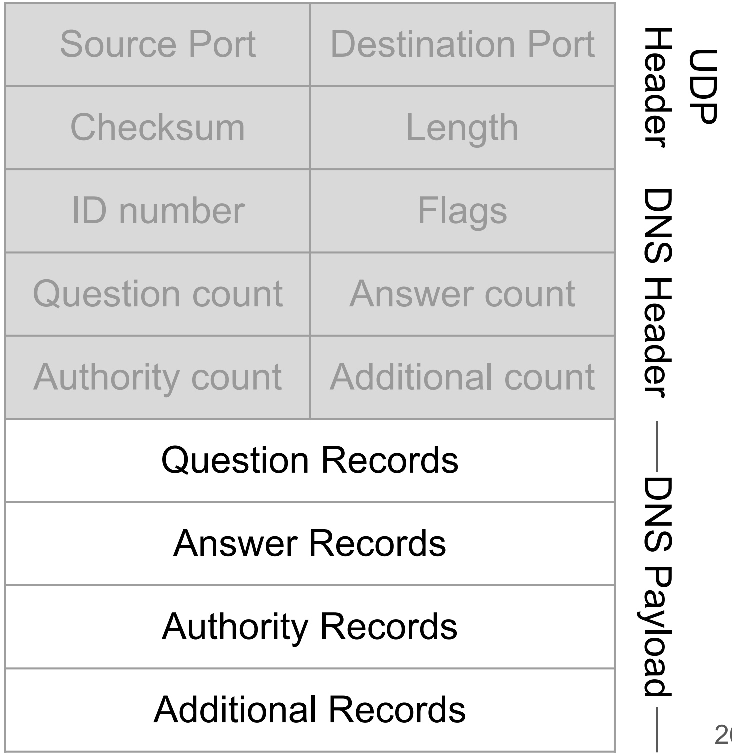
DNS Packet Format: DNS Payload

- **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

Source Port	Destination Port	UDP Header
Checksum	Length	
ID number	Flags	DNS Header
Question count	Answer count	
Authority count	Additional count	
Question Records		DNS Payload
Answer Records		
Authority Records		
Additional Records		

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

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

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)

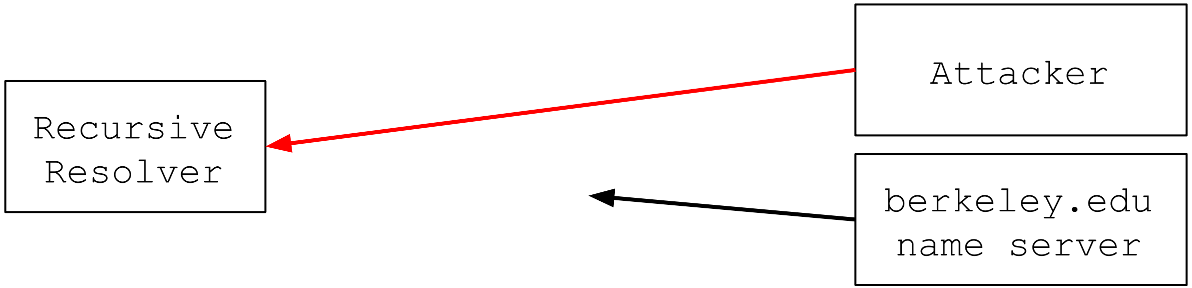
Security Risk: Man-in-the-middle (MITM) Attackers

- 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!



Security Risk: Off-Path Attackers

- The off-path attacker needs to guess the ID field to spoof a response
  - If the ID in the response doesn't match the ID in the request, the resolver won't accept the response
- If the ID number is randomly generated:
  - Probability of guessing correctly = $1/2^{16}$
  - Recall: The ID number is 16 bits long
  - Requires approximately 65,000 tries to successfully send a spoofed packet
  - This is too small!

Source Port	Destination Port	UDP Header
Checksum	Length	
ID number	Flags	DNS Header
Question count	Answer count	
Authority count	Additional count	
Question Records		DNS Payload
Answer Records		
Authority Records		
Additional Records		

Security Risk: Off-Path Attackers

- What if the ID field is incremented by 1 for every request?
- Off-path attacker can spoof a packet as follows:
    - Trick the victim into visiting the attacker's website
    - Include this HTML on the attacker's website: `<img src="http://www.attacker.com">`
    - The victim's browser will make a DNS query for `www.attacker.com`
    - If the attacker controls the `attacker.com` DNS name server, they can see the request and learn the ID field
    - Include this HTML on the attacker's website: `<img src="http://www.google.com">`
    - The victim's browser will make a DNS query for `www.google.com`
    - The attacker knows the ID is 1 more than the previous ID, so they can spoof a response!
- **ID numbers need to be random in DNS requests**

Kaminsky Attack

- Notice: If the attacker places **`<img src="http://www.google.com">`** multiple times on their website, the browser will only make 1 DNS query
  - The browser caches address of **`www.google.com`**
  - The attacker only gets one try
- Dan Kaminsky, security researcher, noticed that DNS clients would cache additional glue records as if they were authoritative answers, even though they aren't

Kaminsky Attack

- Now, the attacker can gain more tries at once:
  - The attacker includes
    - `<img src="http://fake1.google.com">`
    - `<img src="http://fake2.google.com">`
    - `<img src="http://fake3.google.com">`
    - `<img src="http://fake4.google.com">`
  - For each, the client makes a request for the domain name
  - The attacker's spoofed response contains:
    - Authority: `fake1.google.com.   172800   IN   NS   www.google.com.`
    - Additional: `www.google.com.   172800   IN   A   6.6.6.6`
  - The client now caches the record for `www.google.com`, and the cache is poisoned!
- See here for draft extra slides.

Defense: Source Port Randomization

- Randomize the source port of the DNS query
  - The attacker must guess the destination port of the response in addition to the query ID
  - This adds 16 bits to guess, to total 2^{32} possibilities
- Other ways to increase entropy:
  - Randomly capitalize the domain, since the question is copied in the response

Source Port	Destination Port	UDP Header
Checksum	Length	
ID number	Flags	DNS Header
Question count	Answer count	
Authority count	Additional count	
Question Records		DNS Payload
Answer Records		
Authority Records		
Additional Records		

Defense: Glue Validation

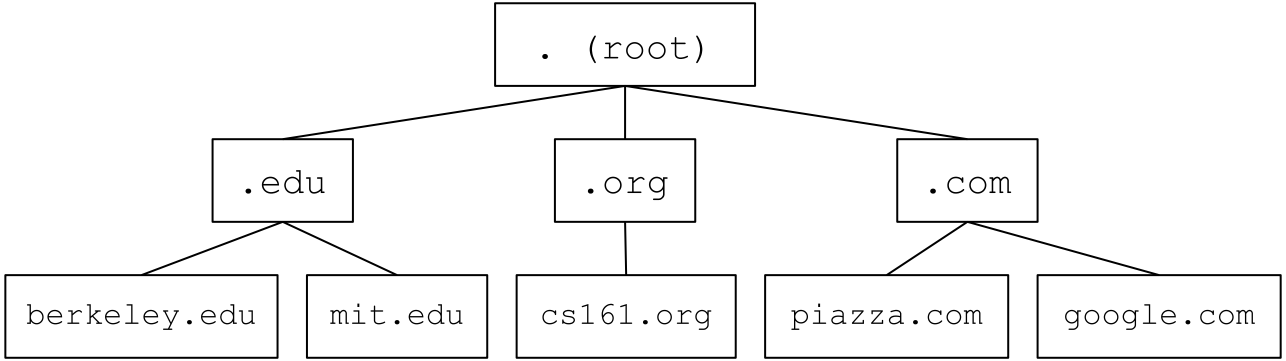
- **Don’t cache glue records as part of DNS lookups**
  - They are necessary, since NS records are given in terms of domain names, not IP addresses
  - If you want to cache, you can perform a separate recursive DNS lookup to validate the glue record authoritatively
- Issue: This was not implemented by all DNS software
  - Unbound, a major DNS implementation, implemented validation
  - BIND, the oldest and most common implementation, did not
    - Mainly for political reasons: They supported DNSSEC, which uses cryptography to validate DNS records (we’ll look at this next time)

Profiting from DNS Attacks

- Suppose you take over a lot of home routers… How do you make money from your attack?
- Change the DNS server settings
  - Each router is programmed with the IP address of the recursive resolver
  - Replace the IP address of the recursive resolver with the attacker's IP address
  - Cache poisoning attacks are now possible!
- Redirect all DNS requests for ads to an attacker-controlled domain and serve attacker-chosen ads to the victim
  - The attacker can now sell this advertising space!
- TLS can defend against this (recall: end-to-end security)

DNS: Summary

- DNS (Domain Name System): An Internet protocol for translating human-readable domain names to IP addresses
  - DNS name servers on the Internet provide answers to DNS queries
  - Name servers are arranged in a domain hierarchy tree
  - Lookups proceed down the domain tree: name servers will direct you down the tree until you receive an answer
  - The stub resolver tells the recursive resolver to perform the lookup



DNS: Summary

- DNS message structure
  - DNS uses UDP for efficiency
  - DNS packets include a random 16-bit ID field to match requests to responses
  - Data is encoded in records, which are name-value pairs 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
  - Records are separated into four sections
    - Question: Contains query
    - Answer: Contains direct answer to query
    - Authority: Directs the resolver to the next name server
    - Additional: Provides extra information (e.g. the location of the next name server)
  - Resolvers cache as many records as possible (until their time-to-live expires)

DNS Security: Summary

- Cache poisoning attack: Send a malicious record to the resolver, which caches the record
  - Causes packets to be sent to the wrong place (e.g. to the attacker, who becomes a MITM)
- Risk: Malicious name servers
  - Defense: Bailiwick checking: Resolver only accepts records in the name server's zone
- Risk: Network attackers
  - MITM attackers can poison the cache without detection
  - On-path attackers can race the legitimate response to poison the cache
  - Off-path attackers must guess the ID field (Defense: Make the ID field random)
    - Kaminsky attack: Query non-existent domains and put the poisoned record in the additional section (which will still be cached). Lets the off-path attacker try repeatedly until succeeding
    - Defense: Source port randomization (more bits for the off-path attacker to guess)