Network Security / 32. DNS

# 32. DNS

The Internet is commonly indexed in two different ways. Humans refer to websites using human-readable names such as `google.com` and `eecs.berkeley.edu`, while computers refer to websites using IP addresses such as `172.217.4.174` and `23.195.69.108`. **DNS**, or the **Domain Name System**, is the protocol that translates between the two.

## 32.1. Name servers

It would be great if there was single server that stored a mapping from every domain to every IP address that everyone could query, but unfortunately, there is no server big enough to store the IP address of every domain on the Internet and fast enough to handle the volume of DNS requests generated by the entire world. Instead, DNS uses a collection of many **name servers**, which are servers dedicated to replying to DNS requests.

Each name server is responsible for a specific zone of domains, so that no single server needs to store every domain on the Internet. For example, a name server responsible for the `.com` zone only needs to answer queries for domains that end in `.com`. This name server doesn't need to store any DNS information related to `wikipedia.org`. Likewise, a name server responsible for the `berkeley.edu` zone doesn't need to store any DNS information related to `stanford.edu`.

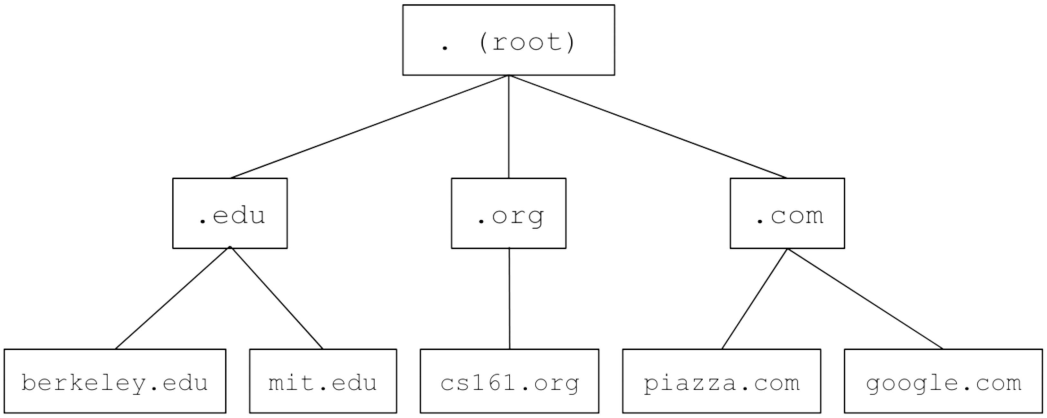
Even though it has a special purpose (responding to DNS requests), a name server is just like any other server you can contact on the Internet–each one has a human-readable domain name (e.g. `a.edu-servers.net`) and a computer-readable IP address (e.g. `192.5.6.30`). Be careful not to confuse the domain name with the zone. For example, this name server has `.net` in its domain, but it responds to DNS requests for `.edu` domains.

## 32.2. Name server hierarchy

You might notice two problems with this design. First, the `.com` zone may be smaller than the entire Internet, but it is still impractical for one name server to store all domains ending in `.com`. Second, if there are many name servers, how does your computer know which one to contact?

DNS solves both of these problems by introducing a new idea: when you query a name server, instead of always returning the IP address of the domain you queried, the name server can also direct you to another name server for the answer. This allows name servers with large zones such as `.edu` to redirect your query to other name servers with smaller zones such as `berkeley.edu`. Now, the name server for the `.edu` zone doesn't need to store any information about `eecs.berkeley.edu`, `math.berkeley.edu`, etc. Instead, the `.edu` name server stores information about the `berkeley.edu` name server and redirects requests for `eecs.berkeley.edu`, `math.berkeley.edu`, etc. to a `berkeley.edu` name server.

DNS arranges all the name servers in a tree hierarchy based on their zones:

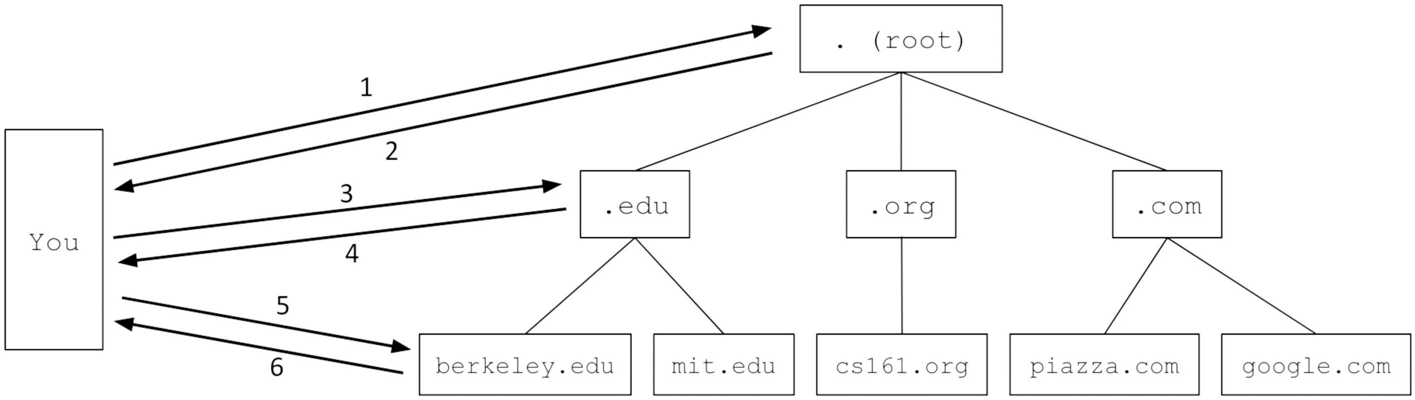


The **root server** at the top level of the tree has all domains in its zone (this zone is usually written as `.`). Name servers at lower levels of the tree have smaller, more specific zones. Each name server is only responsible for storing information about their children, except for the name servers at the bottom of the tree, which are responsible for storing the actual mappings from domain names to IP addresses.

DNS queries always start at the root. The root will direct your query to one of its children name servers. Then you make a query to the child name server, and that name server redirects you to one of its children. The process repeats until you make a query to a name server at the bottom of the tree, which will return the IP address corresponding to your domain.

To redirect you to a child name server, the parent name server must provide the child's zone, human-readable domain name, and IP address, so that you can contact that child name server for more information.

As an example, a DNS query for `eecs.berkeley.edu` might have the following steps. (A comic version of this query is available at https://howdns.works/.)



1 You to the root name server: Please tell me the IP address of `eecs.berkeley.edu`.

2 Root server to you: I don't know, but I can redirect you to another name server with more information. This name server is responsible for the `.edu` zone. It has human-readable domain name `a.edu-servers.net` and IP address `192.5.6.30`.

3 You to the `.edu` name server: Please tell me the IP address of `eecs.berkeley.edu`.

4 The `.edu` name server to you: I don't know, but I can redirect you to another name server with more information. This name server is responsible for the `berkeley.edu` zone. It has human-readable domain name `adns1.berkeley.edu` and IP address `128.32.136.3`.

5 You to the `berkeley.edu` name server: Please tell me the IP address of `eecs.berkeley.edu`.

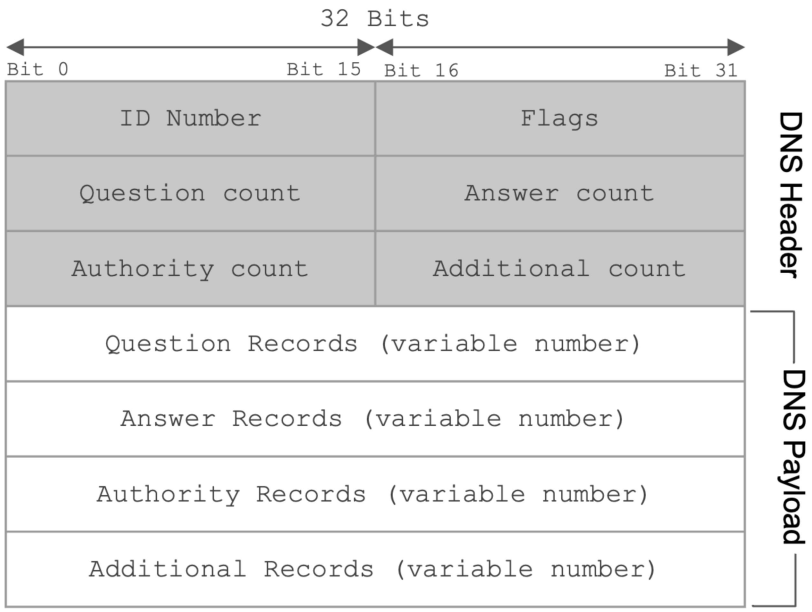
6 The `berkeley.edu` name server to you: OK, the IP address of `eecs.berkeley.edu` is `23.185.0.1`.

A note on who is actually sending the DNS queries in this example: Your computer can manually perform DNS lookups, but in practice, your local computer usually delegates the task of DNS lookups to a **DNS Recursive Resolver** provided by your Internet service provider (ISP), which sends the queries, processes the responses, and maintains an internal cache of records. When performing a lookup, the **DNS Stub Resolver** on your computer sends a query to the recursive resolver, lets it do all the work, and receives the response. When thinking about DNS requests, you can usually focus on the messages being sent between the recursive resolver and the name server.

Congratulations, you now understand how DNS translates domains to IP addresses! The rest of this section describes the specific implementation details of DNS.

## 32.3 DNS Message Format

Since every website lookup must start with a DNS query, DNS is designed to be very lightweight and fast - it uses UDP (best-effort packets, no TCP handshakes) and has a fairly simple message format.


32 Bits
Bit 0
Bit 15
Bit 16
Bit 31
ID Number
Flags
Question count
Answer count
Authority count
Additional count
Question Records (variable number)
Answer Records (variable number)
Authority Records (variable number)
Additional Records (variable number)
DNS Header
DNS Payload

The first field is a 16 bit **identification field** that is randomly selected per query and used to match requests to responses. When a DNS query is sent, the ID field is filled with random bits. Since UDP is stateless, the DNS response must send back the same bits in the ID field so that the original query sender knows which DNS query the response corresponds to.

Sanity check: Which type(s) of adversary can read this ID field? Which type(s) of adversary cannot read the ID field and must guess it when attacking DNS?[1]

The next 16 bits are reserved for flags, which specify whether the message is a query or a response, as well as whether the query was successful (e.g. the `NOERROR` flag is set in the reply if the query succeeded, the `NXDOMAIN` flag is set in the reply if the query asked about a non-existent name).

The next field specifies the number of questions asked (in practice, this is always 1). The three fields after that are used in response messages and specify the number of **resource records** (RRs) contained in the message. We'll describe each of these categories of RRs in depth later.

The rest of the message contains the actual content of the DNS query/response. This content is always structured as a set of RRs, where each RR is a key-value pair with an associated type.

For completeness, a DNS record key is formally defined as a 3-tuple `<Name, Class, Type>`, where `Name` is the actual key data, `Class` is always `IN` for Internet (except for special queries used to get information about DNS itself), and `Type` specifies the record type. A DNS record value contains `<TTL, Value>`, where `TTL` is the time-to-live (how long, in seconds, the record can be cached), and `Value` is the actual value data.

There are two main types of records in DNS. **A type records** map domains to IP addresses. The key is a domain, and the value is an IP address. **NS type records** map zones to domains. The key is a zone, and the value is a domain.

Important takeaways from this section: Each DNS packet has a 16-bit random ID field, some metadata, and a set of resource records. Each record falls into one of four categories (question, answer, authority, additional), and each record contains a type, a key, and a value. There are A type records and NS type records.

## 32.4. DNS Lookup

Now, let's walk through a real DNS query for the IP address of `eecs.berkeley.edu`. You can try this at home with the `dig` utility–remember to set the `+norecurse` flag so you can unravel the recursion yourself.

Every DNS query begins with the root server. For redundancy, there are actually 13 root servers located around the world. We can look up the IP addresses of the root servers, which are public and well-known. In a real recursive resolver, these addresses are usually hardcoded.

The first root server has domain `a.root-servers.net` and IP address `198.41.0.4`. We can use `dig` to send a DNS request to this address, asking for the IP address of `eecs.berkeley.edu`.

```
$ 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
...
```

In the first section of the answer, we can see the header information, including the ID field (`26114`), the return flags (`NOERROR`), and the number of records returned in each section.

The **question section** contains 1 record (you can verify by seeing `QUERY: 1` in the header). It has key `eecs.berkeley.edu`, type `A`, and a blank value. This represents the domain we queried for (the value is blank because we don't know the corresponding IP address).

The **answer section** is blank (`ANSWER: 0` in the header), because the root server didn't provide a direct answer to our query.

The **authority section** contains 13 records. The first one has key `.edu`, type `NS`, and value `a.edu-servers.net`. This is the root server giving us the zone and the domain name of the next name server we should contact. Each record in this section corresponds to a potential name server we could ask next.

The **additional section** contains 27 records. The first one has key `a.edu-servers.net`, type `A`, and value `192.5.6.30`. This is the root server giving us the IP address of the next name server by mapping a domain from the authority section to an IP address.

Together, the authority section and additional section combined give us the zone, domain name, and IP address of the next name server. This information is spread across two sections to maintain the key-value structure of the DNS message.

For completeness: `172800` is the TTL (time-to-live) for each record, set at 172,800 seconds = 48 hours here. The `IN` is the Internet class and can basically be ignored. Sometimes you will see records of type `AAAA`, which correspond to IPv6 addresses (the usual `A` type records correspond to IPv4 addresses).

Sanity check: What name server do we query next? How do we know where that name server is located? What do we query that name server for?[2]

```
$$ 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 next query also has an empty answer section, with `NS` records in the authority section and `A` records in the additional section which give us the domains and IP addresses of name servers responsible for the `berkeley.edu` zone.

```
$ 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
```

Finally, the last query gives us the IP address corresponding to `eecs.berkeley.edu` in the form of a single `A` type record in the answer section.

In practice, because the recursive resolver caches as many answers as possible, most queries can skip the first few steps and used cached records instead of asking root servers and high-level name servers like `.edu` every time. Caching helps speed up DNS, because fewer packets need to be sent across the network to translate a domain name to an IP address. Caching also helps reduce request load on the highest-level name servers.

## 32.5. DNS Security: Bailiwick

DNS is insecure against a malicious name server. For example, if a `berkeley.edu` name server was taken over by an attacker, it could send answer records that point to malicious IP addresses.

However, a more dangerous exploit is using the additional section to poison the cache with even more malicious IP addresses. For example, this malicious DNS response would cause the resolver to associate `google.com` with an attacker-owned IP address `6.6.6.6`.

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


...
;; ADDITIONAL SECTION:
adns1.berkeley.edu.  172800   IN   A    128.32.136.3
www.google.com       999999   IN   A    6.6.6.6
...
```

To prevent any malicious name server from doing too much damage, resolvers implement **bailiwick checking**. With bailiwick checking, a name server is only allowed to provide records in its zone. This means that the `berkeley.edu` name server can only provide records for domains under `berkeley.edu` (not `stanford.edu`), the `.edu` name server can only provide records for domains under `.edu` (not `google.com`), and the root name servers can provide records for anything.

## 32.6. DNS Security: On-path attackers and off-path attackers

Against an on-path attacker, DNS is completely insecure - everything is sent over plaintext, so an attacker can read the request, construct a malicious response message with malicious records and the correct ID field, and race to send the malicious reply before the legitimate response. If the time-to-live (TTL) of the malicious records is set to a very high number, then the victim will cache those malicous records for a very long time.

For both on-path and off-path attackers, if the legitimate response arrives before the fake response, it is cached. Caching limits the attacker to only a few tries per week, because future requests for that domain can reference the cache, so no DNS queries are sent. Since off-path attackers must guess the ID field with a $1/2^{16}$ probability of success, and they only get a few tries per week, DNS was believed to be secure against off-path attackers, until Dan Kaminsky discovered a flaw in the DNS protocol in 2008. This attack was so severe that Kaminsky was awarded with a Wikipedia article.

## 32.7. DNS Security: Kaminsky attack

The Kaminsky attack relies on querying for nonexistent domains. Remember that the legitimate response for a nonexistent domain is an `NXDOMAIN` status with no other records, which means that nothing is cached! This allows the attacker to repeatedly race until they win, without having to wait for cached records to expire.

An attacker can now include malicious additional records in the fake response for the nonexistent `fake161.berkeley.edu`:

```
$$ dig fake161.berkeley.edu

;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 29439
;; flags: qr aa; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 1

;; QUESTION SECTION:
;fake161.berkeley.edu.        IN  A

;; ADDITIONAL SECTION:
berkeley.edu.     999999    IN  A   6.6.6.6
```

If the fake response arrives first, the resolver will cache the malicious additional record. Notice that this doesn’t violate bailiwick checking, since the name server responsible for answering `fake161.berkeley.edu` can provide a record for `berkeley.edu`.

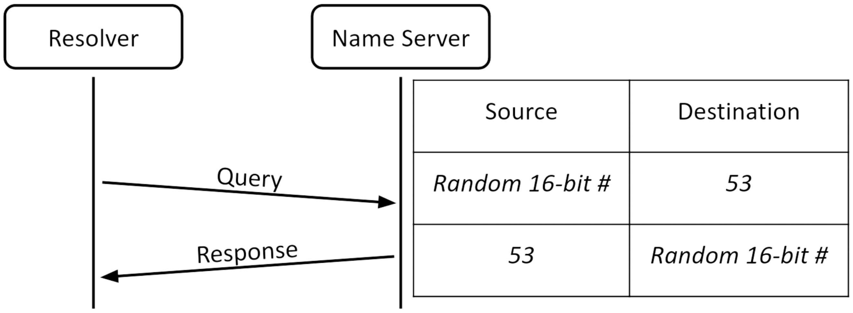
Now that the attacker can try as many times as they want, all that’s left is to force a victim to make thousands of DNS queries for nonexistent domains. This can be achieved by tricking the victim into visiting a website that tries to load lots of nonexistent domains:

```
<img src="http://fake001.berkeley.edu/image.jpg"/>
<img src="http://fake002.berkeley.edu/image.jpg"/>
<img src="http://fake003.berkeley.edu/image.jpg"/>
...
```

This HTML snippet will cause the victim’s browser to try and fetch images from `http://fake001.berkeley.edu/image.jpg`, `http://fake002.berkeley.edu/image.jpg`, etc. To fetch these images, the browser will first make a DNS request for the domains `fake001.berkeley.edu`, `fake002.berkeley.edu`, etc. For each request, if the legitimate response arrives before the malicious response, or if the off-path attacker incorrectly guesses the ID field, nothing is cached, so the attacker can immediately try again when the victim makes the next DNS request to the next non-existent domain.

The Kaminsky attack allows on-path attackers to race until their fake response arrives first and off-path attackers to race until they successfully guess the ID field. There is no way to completely eliminate the Kaminsky attack in regular DNS, although modern DNS protocols add **UDP source port randomization** to make it much harder.

Recall that UDP is a transport-layer protocol like TCP, so a UDP packet requires a source port and destination port. The destination port must be well-known and constant (in practice, it is always 53), so everyone can send UDP packets to the correct port on the name server. However, DNS doesn't specify what source port the resolver uses to send queries, so source port randomization uses a random 16-bit source port for each query. The name server must send the response packet back to the correct source port of the resolver, so it must include the source port number in the destination port field of the response. Now, an attacker must guess the 16-bit ID field and the 16-bit source port in order to successfully forge a response packet. This decreases an off-path attacker's probability of success to $1/2^{32}$, which is much harder, but certainly not impossible.



| Source | Destination |
|---|---|
| *Random 16-bit #* | *53* |
| *53* | *Random 16-bit #* |

Sanity check: How much extra security does source port randomization provide against on-path attackers?[3]

[1] A: MITM and on-path can read the ID field. Off-path must guess the ID field. ↩

[2] Query `a.edu-servers.net`, whose location we know because of the records in the additional section. Query for the IP address of `eecs.berkeley.edu` just like before. ↩

[3] A: None, on-path attackers can see the source port value. ↩