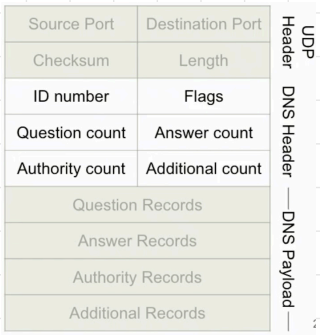
Computers only understand IP addresses, not human readable domain names like www.google.com. Therefore, we need someway to translate between human readable domains and IP addresses.
DNS (Domain Name System): An Internet protocol for translating human-readable domain names to IP addresses.
Name server: A server on the internet responsible for answering DNS requests. We send a DNS query to the name server and the name server sends a DNS response with the answer (e.g. "The IP address of www.google.com is …).
Name Server Hierarchy: If one name server doesn't know the answer to your query, the name server can direct you to another name server. We can do this by arranging the name servers in a tree hierarchy, with the intuition being that name servers will direct you down the tree until you receive the answer to your query. For example the .edu name server is responsible for queries like eecs.berkeley.edu, but not a query like mail.google.com. Each node in the tree represents a zone of domain and can delegate part of its zone to someone else.
DNS Lookup: All DNS queries start with a request to the root name server. We ask the root and the root says that it doesn't know but he has delegated authority to the .edu name server. Then you ask .edu the IP address of eecs.berkeley.edu. .edu doesn't know so it delegates authority to the berkeley.edu name server. Finally, the berkeley.edu name server responds with the IP address of eecs.berkeley.edu.
Stub Resolver: your computer tells another resolver to perform the query for you, and it only contacts the recursive solver and receives the answer. The Recursive Resolver is actually responsible to make the DNS queries and is usually run by ISPs or application providers. Remember that in DHCP, the user needs to fetch the IP address of the DNS server (a recursive resolver) for these queries. The stub resolver sends the query to the recursive resolver, the recursive resolver returns the final answer to the stub resolver. DNS uses UDP because we want DNS to be lightweight and fast.

ID number: Used to associate the queries with responses
Counts: The number of records of each type in the DNS payload
Resource Records: Question, answer, authority, and additional section
A type records: maps a domain name to IPv4
NS type record: designates another DNS server to handle a domain
Question section: What is being asked
Answer section: A direct response to the question
Authority section: A delegation of authority for the question - used to direct the resolver to the next name server and of type NS
Addition section: Additional information to help with the response - also called glue records. Provides non-authoritative records for domains

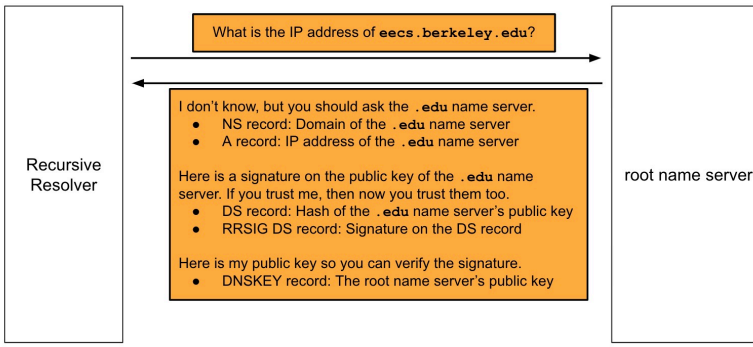


Cache Poisoning: Returning a malicious record to the client. Ex: supply a malicious A record mapping the attacker's IP address to a domain. Malicious name servers can lie and supply a malicious answer.
Defense - Bailiwick Checking: the resolver only accepts records if they are in the name server's zone.
MITM Attackers: they can poison the cache by adding, removing, or changing any record in the DNS response to their will. On-Path Attacker: They can attempt to race the actual name servers response to the recursive solver. Off-path attackers: they have to guess the ID field to spoof the response so they need to guess it. If the ID is randomly generated, the probability of guessing correctly is 1/2^16.
Kaminsky Attack: DNS clients would cache glue records as if they were authoritative answer. Use fake website with guessed ID fields and fake authority sections linking the search domain name with Mal's IP address

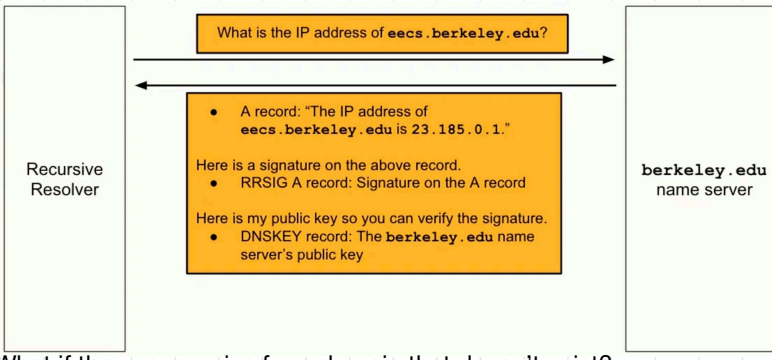
	Attacker **correctly** guesses the ID number	Attacker does **not** correctly guess the ID number
Attacker **beats** the race condition against the legitimate NS	The recursive resolver caches a mapping from legitimate domain names to the attacker's desired IP addresses.	The recursive resolver ignores the response because the ID does not match the request sent earlier. The recursive resolver caches something saying "This domain does not exist". Try again with another fake domain!
Attacker does **not** beat the race condition against the legitimate NS	The recursive resolver caches something saying "This domain does not exist". Try again with another fake domain!	The recursive resolver caches something saying "This domain does not exist". Try again with another fake domain!

Defense - Source Port Randomization: Randomize the source port of the DNS query to increase the number of guesses needed.
Defense - Glue Validation: Don't cache glue records as part of DNS lookups.
DNSSEC:
Public-Key Infrastructure: Name servers are arranged in a hierarchy, as in ordinary DNS. Parents can delegate trust to children. The parent signs the child's public key to delegate trust to the child - if you trust the parent name server, then now you trust the child name server. We implicitly trust the root name server. This method defeats malicious name servers.
A DNS record has a name, type, and value. A group of DNS records with the same name and type form a resource record set (RRSET). Used for simplifying signatures - instead of signing every record separately, we can sign an entire RRSET at once.
RRSIG (resource record signature): encodes signatures on records
DNSKEY: encodes the public key
DS (delegated signer): encodes the child's public key (used to delegate trust (the parent signs the child's public key to delegate trust to the child - DNSSEC delegates trust with two records: a DS type record with the hash of the signers name and the child's PK and an RRSIG type record with a signature on the DS record.

Query:



Final Answer:



What if the user queries for a domain that doesn't exist?
Option 1: Don't authenticate NXDOMAIN requests. Issue is that if NXDOMAIN responses don't have to be signed, attacker can spoof NXDOMAIN response for DoS
Option 2: Keep the private key in the name server itself, so it signs NXDOMAIN responses. Issues is that name servers have access to private key, which is an issue if they are hacked. Signing is also slow. Therefore, we need a way to prove that a domain doesn't exist ahead of time.
To prove no existence of a record type: sign a record stating that no record of a given type exists. To prove nonexistentence of a domain, provide two adjacent domains alphabetically, so that you know that no domain in the middle exists. Issue with this model (called NSEC) is that attacker can find every single subdomain of a domain by sequence of faulty DNS requests. Instead, we should store sequence of adjacent hashes. Still possible to brute-force all reasonable domain names. Only prevents attackers from learning long, random domain names, which would make brute-force difficult. NSEC3: given a nonexistent domain name, generate a signature on the fly that states that no record exists between H(name) - 1 and H(name) + 1. This requires online signature generation. Instead, we use offline signature generation. Offline signature: the application that computes signature is separate from the application that serves the signatures. This is efficient since records are signed ahead of time. Attacker must compromise the signature generation system. If the system is separate from the name server, we must compromise both.