How DNS Works (and How Hackers Poison It)

A complete, plain-language study guide

NetSec Visualized · Companion to Episode 002 · Aligned to Network+ and Security+

DNS is the phone book of the internet: it turns a name like `netsecviz.net` into an IP address. It is also one of the easiest things on the internet to lie to. This guide walks through how a DNS lookup really works, and then how an attacker poisons it to send a whole network to a fake site, no password required. Every term is defined the first time it appears.

The main text explains each idea plainly; the **Go deeper** notes add the detail an engineer or exam expects; the **Common misconception** boxes clear up what people get wrong; and the **What to remember** boxes are your review. The diagrams are taken straight from the episode.

The big picture

You type a name, but computers route by numbers, not names. So before your browser can send anything, it needs the IP address for that name. **DNS** (the Domain Name System) is what turns one into the other. A lookup runs through:

1. **Caches** (browser, then OS): a recent answer beats a fresh lookup.
2. **A recursive resolver**: a server that does the whole lookup for you.
3. **The hierarchy** (root → TLD → authoritative): the walk that finds the real record.
4. **Caching with a TTL**: the answer is stored so the next lookup is instant.

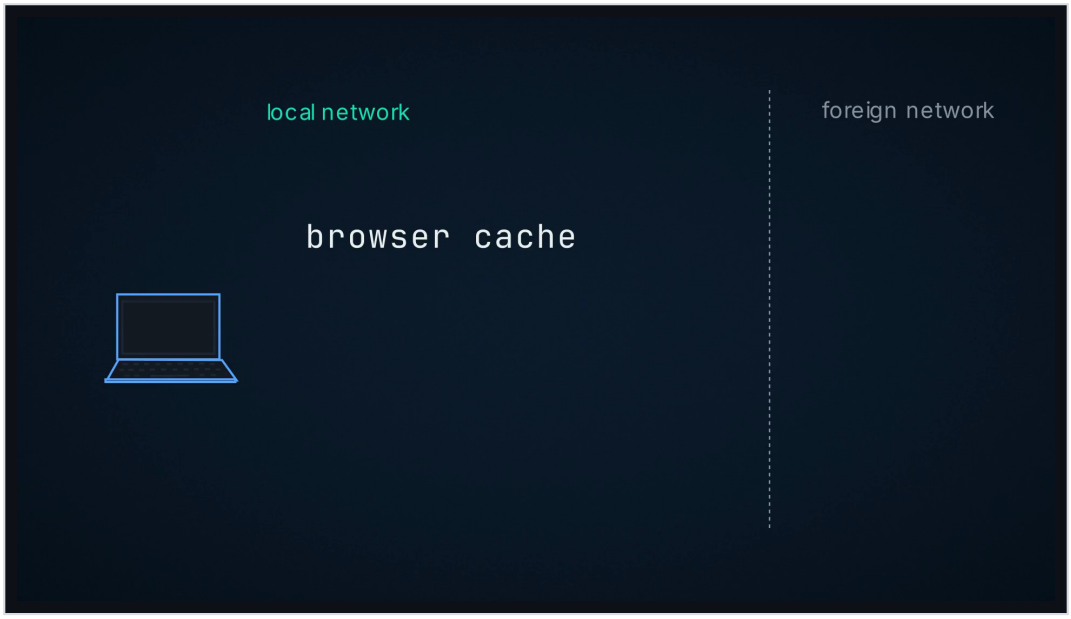
Then the security half: DNS was designed to **trust the first answer that looks right**, and that single assumption is what cache poisoning abuses.

Stage 1 — Your browser checks its caches

In one sentence: *before asking anyone, your browser checks its own memory, because a saved answer is instant.*

The browser keeps a **cache**, a small store of recent lookups. Seen this name recently? Use the saved answer, done. Nothing there? It asks the **operating system**, which keeps its own cache. Only if both miss does it actually go out to the network.

> **WHAT TO REMEMBER**
>
> lookup order is browser cache → OS cache → the network. Caches exist so most lookups never leave your machine.



the browser and OS caches, checked before any network request.

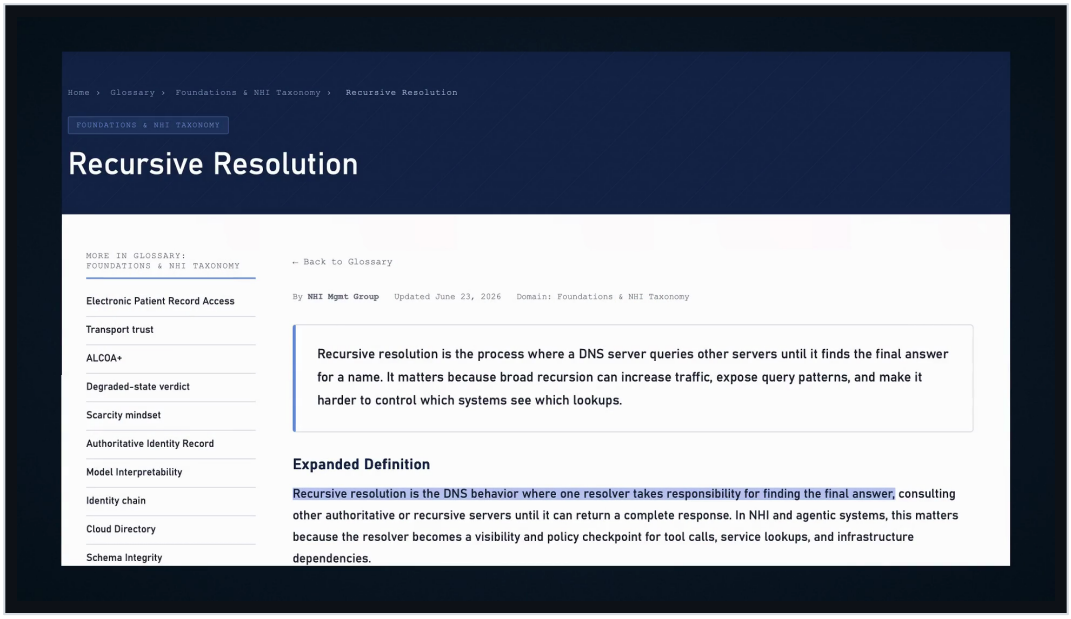
Stage 2 — The recursive resolver

In one sentence: *the first stop on the network is a server that agrees to do the entire lookup for you.*

A **recursive resolver** is a server (usually your ISP's, or a public one like `8.8.8.8`) that takes responsibility for finding the final answer and calling you back. You ask it once; it does the legwork.

> **GO DEEPER**
>
> "recursive" describes the service the resolver gives you (it chases the whole chain on your behalf). The resolver's own walk down the hierarchy is technically **iterative**: each server it asks refers it to the next, rather than answering fully.

> **WHAT TO REMEMBER**
>
> a recursive resolver does the full lookup for you and returns one final answer. `8.8.8.8` is Google's public resolver.



the recursive resolver taking the request from the browser.

Stage 3 — The hierarchy: root, TLD, authoritative

In one sentence: *if the resolver has never heard of the name, it climbs a chain of servers to find who holds the real record.*

The resolver walks three tiers:

1. A **root server**, the top of the tree. It does not know the answer, but it knows who runs `.net`.
2. A **TLD server** (Top-Level Domain, here for `.net`). It does not know the exact answer, but it knows who is **authoritative** for the name.
3. The **authoritative server**, the one server that holds the real record. It hands back the IP.

Root → TLD → authoritative. Three hops, and the name is now a number.

> **GO DEEPER**
>
> DNS answers come as **records**. An **A record** maps a name to an IPv4 address, an **AAAA record** to IPv6, and a **CNAME** points one name at another. The authoritative server is the source of truth for a domain's records.

> **WHAT TO REMEMBER**
>
> root points to TLD, TLD points to authoritative, authoritative holds the real record. It is a referral chain, not one big lookup.

the resolver walking root → TLD .net → authoritative for netsecviz.net.

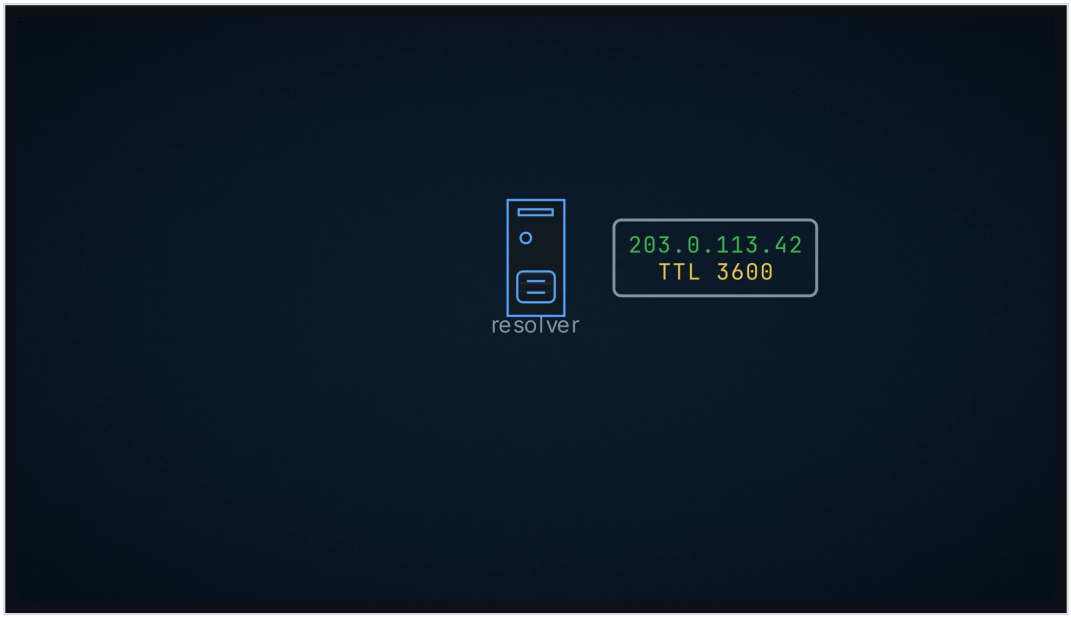
Stage 4 — Caching and TTL

In one sentence: *the resolver saves each answer for a set time, so the next person who asks gets it instantly.*

The resolver does not throw the answer away. It caches it for a window called the **TTL** (Time To Live), a timer set by the domain's owner. So the next person who asks for that name skips the entire walk. Most lookups are answered from a cache and never climb the chain at all.

> **COMMON MISCONCEPTION**
>
> DNS is not looked up fresh every time. Caching at the browser, OS, and resolver means the full root-to-authoritative walk is the exception. This is also why a poisoned answer is so damaging: it sticks in the cache for the whole TTL.

> **WHAT TO REMEMBER**
>
> TTL controls how long a cached answer is reused. Caching makes DNS fast, and makes a poisoned cache entry persist.



the resolver caching the answer with its TTL.

Stage 5 — The trust flaw

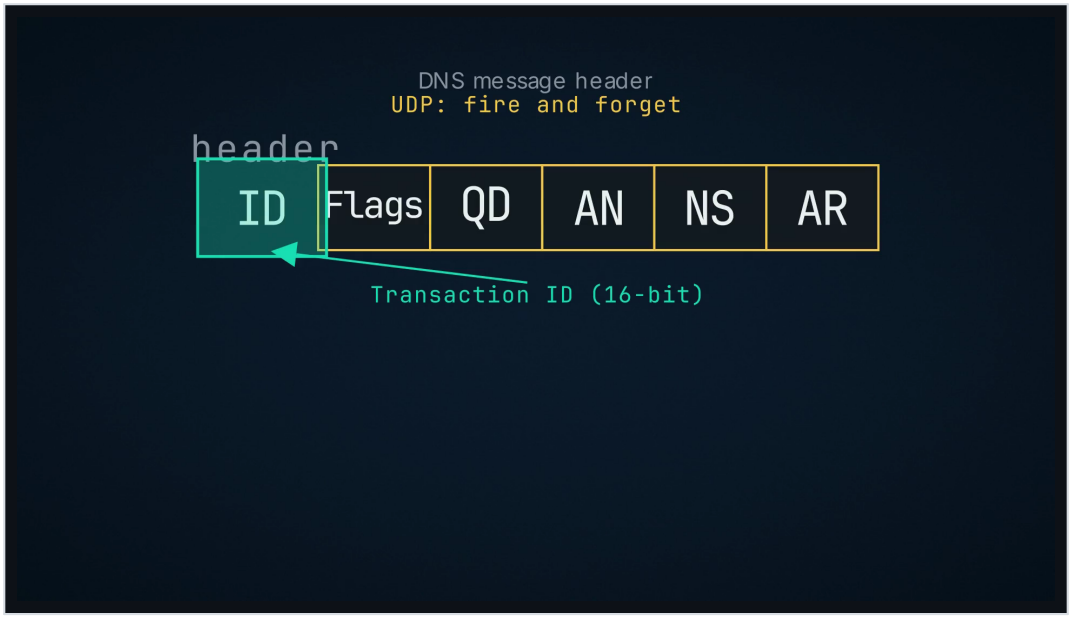
In one sentence: *classic DNS matches a reply to a question with a single 16-bit number, and trusts the first reply that matches.*

For most of its life, DNS was missing one thing: proof. Classic DNS rides on **UDP**, a fast, fire-and-forget way to send packets with no handshake. It matches a reply to your question using a single **16-bit number**, the **transaction ID**. The resolver sends a query, and it trusts the **first reply that comes back with the matching ID**. It does not check who actually sent it.

See the problem? If an attacker can guess that ID and get a fake reply in before the real one, the resolver believes the fake.

> **GO DEEPER**
>
> 16 bits is only 65,536 possible IDs. Combined with a predictable UDP **source port** (older resolvers reused one), the space an attacker had to guess was small enough to brute force.

> **WHAT TO REMEMBER**
>
> DNS over UDP authenticates a reply only by a 16-bit transaction ID and the port. Guess both, arrive first, and you are believed. There is no signature by default.



the DNS message header with the 16-bit Transaction ID highlighted.

Stage 6 — Cache poisoning and the Kaminsky attack

In one sentence: *the attacker races the real server with forged replies until one lands in the cache, then everyone gets sent to the wrong place.*

Cache poisoning is a **race**. The attacker floods the resolver with forged replies, each guessing the transaction ID, trying to beat the real authoritative server to the punch. Win the race, and the resolver **caches the lie**. Every person using that resolver is then silently sent to the attacker's server until the TTL expires. No password is stolen and no malware is needed, just the wrong address, trusted completely.

In **2008**, researcher **Dan Kaminsky** made this reliable (**CVE-2008-1447**, the "Kaminsky bug"). His insight: instead of fighting over one already-cached name, have the resolver look up random, made-up subdomains over and over. Each one is a fresh, uncached query, so the attacker gets **unlimited fresh races** until one lands, and the forged reply can include a poisoned record for the whole domain.

> **COMMON MISCONCEPTION**
>
> the attacker does not "hack" the DNS server. They trick the resolver into caching a forged answer. The protocol's trust is the vulnerability, not a bug in one server.

> **WHAT TO REMEMBER**
>
> cache poisoning wins a race to inject a forged reply. Kaminsky's trick (random subdomains) removed the wait for a cache entry to expire, making it fast and reliable. A poisoned entry lasts for its TTL.

the attacker racing forged replies; the poisoned cache entry pointing everyone to the attacker.

Stage 7 — The defenses

In one sentence: *add randomness so the guess is astronomically hard, and sign the records so forgeries fail outright.*

Two fixes:

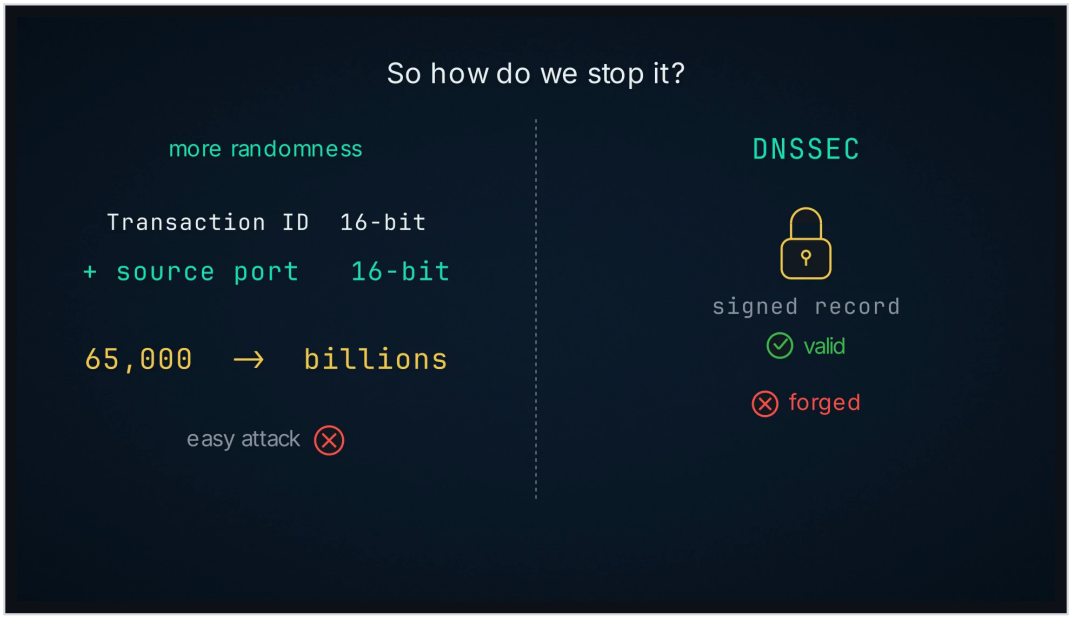
1. **Source-port randomization.** Modern resolvers randomize the UDP source port too, so an attacker must guess the 16-bit transaction ID **and** a 16-bit port. That turns roughly 65,000 guesses into billions, which killed the easy version of the attack.

2. DNSSEC (DNS Security Extensions). It **cryptographically signs** DNS records, so a forged answer simply fails the signature check and is thrown out. This is the only complete fix.

The catch: DNSSEC adoption is still low. Not every domain signs its records, and not every resolver validates signatures, so the protection only holds when both sides participate.

> **GO DEEPER**
>
> port randomization raises the cost of the attack but does not make it impossible; later research (for example the 2020 SADDNS attack) found side channels to recover the port on some setups. DNSSEC is the durable answer because it removes the guessing game entirely: a record is either correctly signed or rejected.

> **WHAT TO REMEMBER**
>
> source-port randomization made poisoning far harder; DNSSEC (signed records) makes it fail by design. DNSSEC is the real fix, but adoption is partial.



the two fixes, added randomness and DNSSEC signed records.

Glossary

- **A / AAAA record:** DNS records mapping a name to an IPv4 (A) or IPv6 (AAAA) address.
- **Authoritative server:** the server that holds the real DNS records for a domain.
- **Cache:** a stored copy of a recent answer, reused to skip repeated work.
- **CNAME:** a DNS record that points one name to another name.
- **CVE-2008-1447:** the "Kaminsky bug," the 2008 DNS cache-poisoning vulnerability.
- **Cache poisoning / DNS spoofing:** injecting a forged answer into a resolver's cache so it returns a malicious IP.
- **DNS (Domain Name System):** the system that maps names to IP addresses.
- **DNSSEC:** DNS Security Extensions, which cryptographically sign records so forgeries are rejected.
- **IP address:** the numeric address of a machine on a network.
- **Iterative vs recursive:** the resolver's walk is iterative (referrals); the service it gives you is recursive (it chases the whole chain).
- **Recursive resolver:** a server that performs the full lookup on your behalf.
- **Root server:** the top of the DNS hierarchy; points to the right TLD server.
- **Source port:** the UDP port a query is sent from; randomizing it adds entropy an attacker must guess.
- **TLD server:** the DNS server for a top-level domain like `.net`.
- **Transaction ID:** the 16-bit number that matches a DNS reply to its query.
- **TTL (Time To Live):** how long a cached DNS answer stays valid.
- **UDP:** a fast, connectionless transport with no handshake, which classic DNS uses.

Quick self-test

1. Put the DNS lookup locations in order, from first checked to last.
2. What is the difference between a recursive resolver and an authoritative server?
3. Name the three tiers of the DNS hierarchy in order.
4. What single value does classic DNS use to match a reply to a query, and how many bits is it?
5. Explain cache poisoning as a "race." What did Kaminsky add to make it reliable?
6. Why does a poisoned answer keep affecting users after the attack stops?
7. Name the two defenses and say which one is the complete fix, and why.

(Answers are in the "What to remember" and "Common misconception" boxes above.)

Further reading

Facts trace to the episode source list (`01-research/sources.md`): Cloudflare Learning and showdns (DNS resolution), the ISC advisory for CVE-2008-1447, and the arXiv survey on DNS cache-poisoning history and mitigation. For exams, map this to Network+ (DNS, record types, the resolution process) and Security+ (DNS attacks and mitigations: spoofing/poisoning, DNSSEC).

NetSec Visualized — networking and security, made visual, and shown how it breaks.