Lecture 8

Gidon Rosalki

2025-12-07

Notice: If you find any mistakes, please open an issue at `https://github.com/robomarvin1501/notes_networking`

1 DNS

1.1 Security - vulnerabilities and solutions

DNS is massively important. Without it, you cannot access any websites. Furthermore, your online identity is inextricably connected to your email. Someone could hijack the DNS for your mail server, and pretend to be you. DNS is the root of trust for the web, when navigating to one's bank website, one expects to be taken to said website, but if the DNS record is compromised, a bad actor could phish all their information.

A first major problem we may discuss is DoS, or Denial of Service. Here, one floods DNS servers with request until they fail. In 2002 and 2015 there were massive Distributed DoS, or DDoS attacks against the root name servers. However, thanks to caches, most users did not even notice, since the root zone file is cached almost everywhere. More targeted attacks can be more effective though, blocking off a local DNS server means that that locality cannot access DNS, and blocking an authoritative server would mean that that domain is inaccessible.

When sitting at a café, and surfing the web, ones laptop can access a site like google.com by asking the local name server. This is run by the café (or their contractor), and can return you *any answer that they please.* This can encourage **man in the middle attacks** (MITM), where there is a site that forwards your query to Google, and forwards back the reply to you, changing anything they like in either direction. You cannot know that you are getting the correct data (though using TLS enabled websites, those that run HTTPS helps prevent this).

This issue is made worse by caching. DNS responses are cached. This is great since it results in a quick response for repeated translations (for both the addresses, and the servers). However, even negative records are cached, since this saves time on nonexistent sites (humans make spelling mistakes). Fortunately, the cached data does eventually time out, but this is usually on the order of magnitude of 10s of minutes.

We have a 3rd problem, with a really metal name of **DNS Cache Poisoning**. Here what happens is a bad actor manages to respond to a query from a resolver, earlier than the actual DNS server. The resolver then caches the wrong result, and until the TTL expires, *every* request for this domain will receive the response of the bad actor, which has been cached. This is significantly worse than spoofing, or MITM attacks, since it can impact everyone that uses the same DNS resolver. This can be the order of an entire ISP (or even more).

So, for DNS poisoning, an attacker wants his IP address returned for a DNS query. When the resolver asks the ns1.google.com for www.google.com, the attack could simply reply first with his own IP. So, we are left with the question of why we do not see this more frequently, and how this is prevented. Firstly, we have transaction IDs: each DNS request contains a 16 bit random number, called the transaction ID. Since the adversary is normally located outside of the message route, then the adversary will not know the transaction ID, and would have to guess, with a success rate of $\frac{1}{2^{16}}$, also known as: very unlikely. Early versions of DNS deterministically incremented the ID field. This made it easy to fake, since managing to observe a single request could mean that the adversary would be able to interfere with all the future requests. Randomisation helped this, but there were also issues of using weak random number generators, so one could predict an ID from seeing a few in the past. Additionally, if a resolver sends many messages requesting the same domain, and the adversary sends many responses, then thanks to the birthday paradox, the success probability can be close to 1.

This brings us to the **Kaminsky attack**, first introduced in 2008, resulting in a great deal of patching, that could only go so far in reducing the attack's likelihood of working. Here, the adversary does not need to wait to try again. The adversary makes a request of the resolver for a site, such as google.com. He most likely loses the race, and any further attempts will be suppressed by the TTL. He may now begin iterating through 1.google.com, 2.google.com, and so on. Each one has a low probability of success, but he may well eventually succeed, say at 185.google.com. So now 185.google.com has been poisoned, but this does not appear to matter much. However, it needs to be noted that this is a *subdomain* of google.com. Subdomains almost always have the same location as the parent domain, so when the TTL runs out, then the server may well respond with the poisoned address of 185.google.com for the queries to google.com.

This is bad. There is no need to wait for the old, good, cached entries to expire, and there is no wait penalty for failing. The attack is only limited by the bandwidth, which is often very large. One can defend against this attack by randomising the UDP port used to respond to the DNS query, so the attacker has to guess that port correctly as well. This is merely a mitigation, and not a proper defence.

The long term solution to these issues is to use DNSSEC, where one adds security keys, and signatures to each response. However, this has been “on the cusp of happening” for a while now, to little effect. Perhaps in a few years it shall become the new standard. Additionally, even if the DNS server supports DNSSEC, if the resolver does not, then it will fall back to previous DNS. The roots adopted DNSSEC in 2010, and in 2016 a study by Chung et al found that 90% of TLDs registered public keys, and 83% of the recursive resolvers support DNSSEC. However, only around 1% of level 2 domains registered public keys. In 2021 almost all the TLDs registered public keys, and in 2025 we have reached around 93%.

2 ARP

This may also be considered L2 vs L3, or when we want to translate between IP addresses, and MAC addresses. ARP stands for Address Resolution Protocol, and was discussed at length in the tutorial (though it will also be discussed now).

Let us suppose that we have some devices connected to a LAN. In order to send messages between them, we need to know the MAC addresses, but have only been informed of their IP addresses. Each node (host, router) on the LAN has an ARP table. This ARP table contains IP and MAC address mappings, along with a TTL field (typically 20 minutes). Whenever a host wants to send a packet, it consults the table, and maps the destination IP address, to the destination MAC address. This neglects when the IP is not in the table. AT this point the sender broadcasts a query for who has the IP address, and the receiver (owner of that IP) responds with its MAC address. At this point, the sender caches the result in its ARP table, and there is no need for a network administrator to get involved.

Now, we also need to route data across LANs, connected by a router. If the host A, knows the destination B’s IP address, but it cannot directly send it a datagram, what happens instead is that A uses ARP to get the MAC of the router’s NIC. It then creates a link-layer frame, with R’s MAC address as the destination, and the frame contains the IP datagram from A to B. When R receives, and decodes the IP datagram, it sees that it is destined for B, and so uses ARP to get B’s MAC address, and creates a frame containing the A to B IP datagram, which is then sent to B.

3 IP address shortage

IPv4 can only handle 32 bit addresses. This is about $2^{32} \approx 4.2 \times 10^9$ devices, which is not many when we consider that there are an excess 8 billion humans. It is not even enough devices for everyone to have a unique address. This has resolutions through processes like Network Address Translation (NAT), where we have local addresses on networks, and only the router has an external IP address, but this is still pushing the problem away. To resolve this we created IPv6. There were also additional motivations, like an improved header format to help speed processing / forwarding up, changing the header to facilitate Quality of Service (QoS) levels, and a fixed 40B header.

However, transitioning from IPv4 to IPv6 is not necessarily quick or easy. Not all router can be upgraded simultaneously, so how can the network operate with a mixed collection of IPv4 and IPv6 routers? In short, it doesn’t very well. We startedd trying to introduce IPv6 just before 2010, and we have now almost reached 50% adoption.