Seven Deadly sins of DNS

September 2018

Seven Deadly Sins of DNS

DNS is a fundamental building block of the internet. If we were to design the internet from first principles today it would probably look quite different than the layers of security that have been added as afterthoughts. We started out with a simple invention-happy mechanism of resolving names to numbers so that the internet could be understood and used with letters and words rather than binary or the decimal number system.

As a result, we have an intersection of (1) the slow and steady layering of of security extensions and (2) backward compatibility. When a bad actor has awareness of security weaknesses and times an attack to utilize the the fact that security mechanisms are not applied universally, we experience vulnerabilities. The distributed and backward-compatible Internet always takes time to get everyone onboard with better security.

To that end, any and all of these seven sins will product compromises of varying degrees. History tells us that we don't always act until we've experienced the pain of not acting. However, for those that want to get ahead of the pack, here are future seven deadly sins of DNS.

1. Using public IP DNS servers on internal networks.
    a. If your internal devices are directly querying public servers such as 8.8.8.8 or 9.9.9.9 or 1.1.1.1 or 208.67.222.222 or whatever your favourite DNS server is, you aren't taking advantage of any caching at an internal caching. You also cannot make intelligent use of split-DNS functionality where nas.network.local for example gives you local NAS access. Some endpoint security purists might say that using DNS over HTTPS (DoH) with a public provider is a good idea, but I'd suggest instead that DoH or DNS over TLS should run internally instead in order to win security and privacy for all, not just for one individual endpoint. Furthermore, Internet-direct DNS queries cause your public resolver to receive many queries that are completely irrelevant including local domain queries, Chrome's random DNS queries, non public suffix domains, etc, creating a very leaky situation from a DNS visibility perspective.
2. Attempting public DNS resolution on non public suffix Top Level Domains (TLDs).
    a. Unless you explicitly prevent TLDs which are not part of https://publicsuffix.org/list, tremendous internal network DNS leakage leaves your network to your upstream DNS provider.
3. Rely on a single upstream recursive DNS resolver organization.
    a. 2018 has proven the fragile status of DNS in the world. Relying on a single DNS resolver for authoritative answer is like a once-in-a-lifetime to call your hero but to get the phone number you check with your significant other who calls his or her best friend who knows your hero. You get Hero's phone number and have no alternate source to verify, so you take your chances.
4. Allow it all through.
    a. DNS is often a critical element of a bad actor's strategy, so filtering DNS is an essential step to get early on in the kill chain of a threat.
5. Treat every device with the same DNS policy.

    a. Why should an IoT device have wide open access? It makes it a perfect jump box as we saw with Casino fish tank thermometer issue. Each device should have access only to what it legitimately requires.
6. Hide DNS activity from users.
    a. Empowering end users is the best way to educate on the massive amount of DNS queries executed with every type of app, service or website. Suddenly the cbc.ca dependencies make it look like one of the most suspect news sites in the world.
7. Focus on only one secure aspect of DNS.
    a. DNSSEC, DNS over HTTPS, DNS over TLS, DNScrypt all play uniquely positive roles in securing the most insecure aspect of DNS. Use it all, everywhere you can, all the time.