IETF 119 hackathon: compact denial of existence for DNSSEC

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

First publication of this article on 18 March 2024. Last update on of 22 March 2024

https://www.bortzmeyer.org/hackathon-ietf-119.html

On March 16 and 17 was the IETF hackathon in Brisbane. I worked on a DNSSEC feature called "compact denial of existence", and implemented it successfully (which was easy).

Compact Denial of Existence (CDE) was originally called "black lies". The name was changed, probably because of some typical US issue with every adjective of color, but also because there was a risk of confusion with lying DNS resolvers, which are something quite different. What is its purpose ? For DNSSEC, you have to sign your records. You can do it offline, on a primary name server (the records and the signatures are then transferred to other authoritative name servers), or online, in every authoritative name server. The first method is more secure (you don't need the private key on every authoritative name server) and uses less CPU resources but it is not usable if you have very dynamic content, generated at every request. For this kind of content, you need online signing.

Online signing of records does not raise any particular issue. The difficulty starts with denial of existence, when you want to sign a statement that a name, or a specific record type does not exist. In that case, you have no data to sign. DNSSEC has several ways to deal with that and the simplest is NSEC records. They are records created to say "there is no name between this one and that one". Let's query a root name server to see these NSEC records :

```
% dig +dnssec @d.root-servers.net doesnotexist
...
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 26586
...
;; AUTHORITY SECTION:
doctor. 86400 IN NSEC dog. NS DS RRSIG NSEC
doctor. 86400 IN RRSIG NSEC 8 1 86400 20240330170000 20240317160000 30903 . hZ08T19claSr8ZkU
...
```

You get a NXDOMAIN (No Such Domain), obviously, and a NSEC record (more than one, actually, but I make things simple, but see later), which says that there is no TLD between `.doctor` and `.dog`. This NSEC record is signed, proving that `.doesnotexist` does not exist.

When signing a complete zone, creating the NSEC records is easy. But if we generate DNS content dynamically, it is more tricky. The DNS server, when signing, does not always know the previous name and the next name, it may not have a complete view of the zone. A trick for dynamic signing was named "white lies" and documented in RFC 4470 [1]. The idea is to generate dynamically a NSEC with a previous name and a next name both very close from the requested name. Something like (this is a real domain, you can try it yourself) :

```
% dig +multi +dnssec +noidnout doesnotexist.dyn.bortzmeyer.fr
...
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 61681
...
;; AUTHORITY SECTION:
˜.doesnotexiss˜.dyn.bortzmeyer.fr. 0 IN NSEC doesnotexist!.dyn.bortzmeyer.fr. RRSIG NSEC
˜.doesnotexiss˜.dyn.bortzmeyer.fr. 0 IN RRSIG NSEC 8 5 0 (
...
```

The names `.doesnotexiss .dyn.bortzmeyer.fr` and `doesnotexist!.dyn.bortzmeyer.fr` were choosen to be close enough of the requested name. This is not the exact algorithm of RFC 4470 because I had problems with some resolvers but the general idea is the same. (Also, `+noidnout` was added because dig has some ideas about what is a displayable domain name. There are no IDN involved.) Here is how Drink `<https://framagit.org/bortzmeyer/drink/>`, the dynamic name server used in the hackathon, written in Elixir, did the "white lies" `<https://framagit.org/bortzmeyer/drink/-/blob/master/lib/drink/server.ex?ref_type=heads#L587>` trick :

```
previous = Drink.Dnssec.previous_name(domain)
nsec = %DNS.Resource{class: :in,
                     domain: to_charlist(DomainName.name(previous)),
                     type: @nsec,
     data: Drink.Dnssec.nsec(domain, is_base)}
```

Now, what is the problem with this way of generating dynamic NSEC records ? I said before that you actually need two (and sometimes three) NSEC records because you also need to prove that the name could not have been generated by a wildcard. Each NSEC will require computation for its signature and, since all of this is done in real-time, we wish to minimize the number of computations (and the size of the response). Hence the Compact Denial of Existence (CDE, formerly known as "black lies"). The idea is to have a NSEC record which does not go from "a name just before" to "a name just after" but from "the requested name" to "a name just after". This way, you no longer need to prove there was no wildcard since you do not claim the name does not exist. (And this could be called a lie, but, since it is done by the authoritative name server which, by definition, tells the truth, the term is better left unused.) But it has an important consequence, the response code must now be NOERROR and not NXDOMAIN, obscuring the fact that the name does not "really" exist (something that some people at the IETF did not like). Here is the result, as produced by Cloudflare name servers :

[1] Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc4470.txt`

```
% dig +multi +dnssec @ns4.cloudflare.com doesnotexist.cloudflare.com
...
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 43367
...
;; AUTHORITY SECTION:
doesnotexist.cloudflare.com. 300 IN NSEC \000.doesnotexist.cloudflare.com. RRSIG NSEC TYPE65283
cloudflare.com. 300 IN RRSIG SOA 13 2 300 (
...
```

As you can see, the left-hand part of the record (the "owner name", `doesnotexist.cloudflare.com`) is now set to the requested name. And the return code is NOERROR, with no data returned in the Answer section. The response says that the name exists, but only with RRSIG, NSEC and TYPE65283 - more on this one later - records.

It should be noted that any ordinary, unmodified, validating resolver will have no problem with this response. It will be validated. CDE does not require any change to the resolver. Also, checking tools like DNSviz `<https://dnsviz.net/>` and Zonemaster `<https://zonemaster.fr/>` will see no problem. CDE can be deployed unilaterally, and this is exactly what Cloudflare did.

So, as we saw, CDE is not new, even if it is not described in any RFC. It is already deployed (remember that the Internet is permissionless : you do not need a RFC to implement and deploy a technology). But what IETF is currently working one is precisely a specification of this technique. The project is currently an Internet-Draft, `draft-ietf-dnsop-compact-denial-of-existence`. This draft describes the basic technique for generating the unique NSEC record and adds :

- In the list of record types that are present at the requested name (the "NSEC bitmap"), a pseudo-type NXNAME, signaling that the NOERROR return code is actually a NXDOMAIN. It is not officially allocated today and implementations use a value reserved for experimentations, 65283 (hence the TYPE65283 above). This can be used by resolvers to restore the NXDOMAIN from the response so that their clients will have a proper error code.
- EDNS signaling that the client understands compact answers and can receive a NXDOMAIN, it won't be surprised to see the NSEC claiming that the name does not exist. This signaling is done with a bit called CO (for Compact OK).

During the IETF hackathon, two authoritative name servers were modified to generate CDE :

- adns_server `<https://github.com/shuque/adns_server>`, written in Python.
- Drink `<https://framagit.org/bortzmeyer/drink/>`, written in Elixir. The code is currently in a separate branch, `compact-denial` (if you retrieved the code with git, `git diff -u -r master -r compact-denial` will give you the code written during the hackathon).

In both cases, the work was simple and fast : once you have dynamic signing (it was already in Drink), CDE is simpler than white lies. The two servers also implemented the NXNAME reporting and accept the signaling via the CO bit. Here is the result with Drink (this test domain was created for the hackathon, and no longer works) :

```
% dig +dnssec +multi nonono.ietf.bortzmeyer.fr TXT
...
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 49495
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 0, AUTHORITY: 4, ADDITIONAL: 1
...
nonono.ietf.bortzmeyer.fr. 0 IN NSEC \000.nonono.ietf.bortzmeyer.fr. \
      RRSIG NSEC TYPE65283

nonono.ietf.bortzmeyer.fr. 0 IN RRSIG NSEC 8 4 0 (
                                20240321040000 20240316154000 43304
                                ietf.bortzmeyer.fr. ...
...
```

As of today, no real resolver has been modified to implement NXDOMAIN restoration and/or EDNS wignaling with CO. This is not mandatory, but would be cool. Zonemaster was happy <https://zonemaster.fr/fr/result/503b9f1695ccdfde> and so was DNSviz <https://dnsviz.net/d/ip.ietf.bortzmeyer.fr/Zfoftw/dnssec/> :

As usual, the hackathon pinpointed some issues that were not always noticed before :

- Since the draft says that explicit requests for a NXNAME record are forbidden, which error code should be returned and, if using EDE (Extended DNS Errors, RFC 8914), which EDE ? (Actually, the problem is larger than just CDE since it seems that the replies to requests of pseudo-types - those between 128 and 255 <https://www.iana.org/assignments/dns-parameters/dns-parameters.xml#dns-parameters-4> - are underspecified. May be another work to start.)
- When a DNS client does not set the DO bit, indicating it does not know about DNSSEC, should we return NXDOMAIN, forgetting about CDE, like most implementations do, or should we return NOERROR, as if the DO bit were set ?
- This is not only about the non-DNSSEC clients : some people are not happy that, by conflating NXDOMAIN and NOERROR, we lose information, and that will make debugging more difficult (unless you have a NXDOMAIN-restoring resolver, which does not exist today). It is important to remember that it is not because something is simple to implement and works, that it is a good idea.

Thanks to Shumon Huque for code, conversation and explanations.