IETF 121 hackathon: greasing DNS answers

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

First publication of this article on 10 November 2024

https://www.bortzmeyer.org/hackathon-ietf-121.html

On November 2 and 3 was the IETF hackathon in Dublin. I worked on the greasing of DNS answers from an authoritative name server. What is greasing ? Continue reading.

One of the big technical problems of the Internet is its ossification : software is written by people who did not read the technical standards, or did not understand them, specially software in the middleboxes (load balancers, firewalls, etc). As a result, some things that are possible according to the technical specification are de facto forbidden by broken software. This makes difficult to deploy new things. For instance, TLS 1.3 had to pretend to be 1.2 (and add an extension to say "I am actually 1.3") because too many middleboxes prevented the establishement of TLS sessions if the version was 1.3 (see RFC 8446 [1], section 4.1.2). This problem is widespread in the Internet, specially since there is typically no way to talk to the middlebox software authors and these boxes are popular among managers.

A way to fight ossification is **greasing**. Basically, the idea is to exercise all the features and options of a protocol from day one, not waiting that you really need them. This way, broken software will be detected immediately, not many years after, when it is entrenched. TLS was the first protocol to go that way (see RFC 8701) and it proved effective. QUIC also uses greasing (RFC 9287).

The DNS could benefit from greasing as well, since it is often difficult to deploy new features, because they sometimes break bad software (it was the case with the cookies of RFC 7873). Hence the current Internet Draft `draft-ietf-dnsop-grease`.

OK, so, let's grease the rusted parts of the Internet but where exactly, and how ? DNS servers are basically of two kinds : resolver and authoritative servers. The version -00 of the draft only mentions resolvers because they are in the best place to test greasing and to report what broke. The general idea is that the resolver sends its queries with "unexpected" values (unallocated EDNS options, unallocated EDNS flags, etc, all of them "legal" according to the RFC). If it receives no reply from the authoritative

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc8446.txt`

server (or a bad one such as FORMERR), and, if retrying without greasing work, the resolver knows there is a problem in the path to this authoritative name server and can log it and/or report it (for instance through RFC 9567). The remaining question is : what we can grease ? We need options that are legal to send but new and unexpected. For plain DNS, there is no hope : there is only one remaining (unallocated) bit <https://www.iana.org/assignments/dns-parameters/dns-parameters.xml#dns-parameters-12> in the flags (RFC 1035, section 4.1.1) of the DNS query. So, it means we can grease only with EDNS stuff : EDNS version number <https://www.iana.org/assignments/dns-parameters/dns-parameters.xml#dns-parameters-14>, EDNS options <https://www.iana.org/assignments/dns-parameters/dns-parameters.xml#dns-parameters-11>, and EDNS flags <https://www.iana.org/assignments/dns-parameters/dns-parameters.xml#dns-parameters-13>. For instance, unknown EDNS options are supposed to be ignored (otherwise, it would never be possible to deploy new options). Now, how to choose the unallocated values to send ? TLS decided to reserve ranges of values for which to choose randomly. The risk is that some bad software will treat this range in a special way but, at least, it guarantees there will be no collision with a future allocation.

This is the current version (-00) of the draft. Now, the work at the hackahton. First, I decided to work on an authoritative server. A priori, it is less useful than a resolver, because, unlike the resolver, the authoritative name server cannnot know if its reply was accepted or not, or created problems. But it could be useful on test zones, to see (for instance through the use of RIPE Atlas probes <https://atlas.ripe.net>) if they have resolution issues. The work was done on the software Drink <https://framagit.org/bortzmeyer/drink/>.

First test, sending back in the reply two EDNS records. Sending two OPT records in a response does not seem forbidden by the RFC (which prohibit it only in a query, RFC 6891, section 6.1.1) but dig does not like it :

```
% dig +norec grease.courbu.re SOA @31.133.134.59
;; Warning: Message parser reports malformed message packet.
```

It creates problems with many other programs and it is not clear if it is legitimate so let's stop here.

Second test, sending an EDNS reply with a version number which is higher than the one requested. This is legal, the last paragraph of Section 6.1.3 of RFC 6891 says that a responder can respond with a higher EDNS version than what was requested by the requestor. (And it explains why, and the limits, for instance to keep the same format.) I tried that for DNS greasing and typical resolvers seem to be happy with it. But DNS testing tools (very useful tools, do not forget to tests your zones with them !) disagree. ednscomp <https://ednscomp.isc.org/> says "expect : OPT record with version set to 0" (not greater-or-equal, stricly equal). DNSviz <https://dnsviz.net/> says "The server responded with EDNS version 1 when a request with EDNS version 0 was sent, instead of responding with RCODE BADVERS. See RFC 6891, Sec. 6.1.3." (We obviously do not read this section in the same way. To me, it mentions BADVERS only in a different context.) And Zonemaster <https://zonemaster.net/> also disagrees with me. So, there is a debate : when a responder knows both version 0 and some higher version (say, version 1), can it reply to a EDNS=0 query with a EDNS=1 response ? Can we use that for greasing ?

Less controversial, adding EDNS options and flags. You can see the result here :

https://www.bortzmeyer.org/hackathon-ietf-121.html

```
% dig @192.168.41.237 grease.courbu.re SOA

; <<>> DiG 9.18.28-0ubuntu0.24.04.1-Ubuntu <<>> @192.168.41.237 grease.courbu.re SOA
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 61647
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 1, flags:; MBZ: 0x0072, udp: 1440
; OPT=16282: 58 ("X")
; OPT=17466: 58 58 58 58 58 58 58 58 58 ("XXXXXXXXX")
; OPT=18095: 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 ("XXXXXXXXXXXXXXXXXX")
; OPT=16375: 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 ("XXXXXXXXXXXXXXXX")
; OPT=18: 06 72 65 70 6f 72 74 07 65 78 61 6d 70 6c 65 03 63 6f 6d 00 (".report.example.com.")
; COOKIE: db279863745c8e7198d4274c54233c48 (good)
;; QUESTION SECTION:
;grease.courbu.re. IN SOA

;; ANSWER SECTION:
grease.courbu.re. 0 IN SOA dhcp-863b.meeting.ietf.org. root.invalid. (
2024111007 ; serial
1800       ; refresh (30 minutes)
300        ; retry (5 minutes)
604800     ; expire (1 week)
86400      ; minimum (1 day)
)

;; Query time: 1 msec
;; SERVER: 192.168.41.237#53(192.168.41.237) (UDP)
;; WHEN: Sun Nov 10 07:33:30 GMT 2024
;; MSG SIZE  rcvd: 224
```

Here, the authoritative name server (a recent version of Drink,using the `--greasing` option at startup), sent :

— A EDNS response with version 1 (remember current version is 0),
— Four EDNS options with unallocated codes, with varying length and values (the last two options have allocated codes, even if dig knows only one, these two options are not greasing),
— Unallocated EDNS flags set (the "0x0072").

Apparently, from tests with various resolver software and through RIPE Atlas probes, it does not break anything, thus paving the way for future allocations. Note that option codes, flags and the number of options are choosen at random, following the draft.

If you want to see the changes it required in the name server, this is this pull request <https://framagit.org/bortzmeyer/drink/-/commit/8e78ceba94b9eab7d77efa3352202e99e100085a>.

Thanks to Shumon Huque and Mark Andrews for code, conversation and explanations.