Developing a monitoring plugin for DNS-over-TLS at the IETF hackathon

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

First publication of this article on 27 March 2017

http://www.bortzmeyer.org/monitor-dns-over-tls.html

The weekend of 25-26 march 2017, I participated to the IETF 98 <https://www.ietf.org/meeting/98/index.html> hackathon in Chicago. The project was to develop a monitoring plugin <https://www.monitoring-plugins.org/> for the DNS-over-TLS privacy protocol, standardized in RFC 7858 [1]. This is a small documentation of the result and of the lessons learned.

A bit of background, first. "Monitoring Plugins" <https://www.monitoring-plugins.org/> is project to develop and maintain an excellent suite of testing programs to be used by many monitoring software like Icinga. Using their API was an obvious choice, allowing the plugin to be used in many places. And DNS-over-TLS? It's a way to improve privacy of DNS users by encrypting the DNS traffic (see RFC 7626 for the privacy issues of the DNS). DNS-over-TLS is described in RFC 7858, published less than one year ago. DNS-over-TLS is implemented in many DNS servers (such as Unbound) and there are several public DNS-over-TLS resolvers <https://portal.sinodun.com/wiki/display/TDNS/DNS-over-TLS+test+servers>. All of them are experimental, "best effort" services and thus some monitoring is a good idea, so we can be sure they actually work most of the time. Existing monitoring plugins like check_dig <https://www.monitoring-plugins.org/doc/man/check_dig.html> cannot run with TLS.

The IETF hackathon is intended for development of IETF-related techniques <https://www.ietf.org/hackathon/98-hackathon.html>. A monitoring plugin for this DNS-over-TLS service is a good fit for a hackathon : hard enough to require some work, but small enough to be reasonably completed in one weekend.

I prepared for the hackathon by setting up a Github repository <https://github.com/bortzmeyer/monitor-dns-over-tls> and exploring the various possibilities. I saw two alternatives :

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc7858.txt

— Use Go because it has both a nice DNS library <https://miek.nl/2014/August/16/go-dns-package/> and a good TLS standard package <https://golang.org/pkg/crypto/tls/>. On the other hand, I'm not sure that the Monitoring Plugins project accept plugins written in Go (I did not find precise rules about that). And the command line arguments parsing package of Go may make difficult to follow exactly the rules of the API.
— Use C with the getdns <https://getdnsapi.net/> package, which can do DNS over TLS (and many other things). Because most monitoring plugins are written in C, there was a lot of code to start with.

I choosed C and getdns for two reasons, the availability of getdns developers at the hackathon (that's the good thing with hackathons, working with people who are at the same table), and the problem of retrieving the PKIX certificate. Why did I need this certificate ? Because I wanted to test things that are TLS-specific, such as a nearby expiration, by far the most common problem with TLS servers.

Using Go and the godns library, it is easy to do a DNS-over-TLS request with the `Exchange()` function. It is easy because it hides everything from the programmer. But it is also what makes it unsuitable for my purpose, it hides the TLS details and provides no way to retrieve the certificate. A possible solution would be to use godns only to create and parse DNS messages and to call directly the Go network and TLS libraries to send messages and receive responses. Then, I would have the certificate in the `conn` object. Certainly doable, but more work. So, I used C and getdns.

At first glance, it was not better, getdns <https://getdnsapi.net/> does not give access to the certificate of the TLS connection. But this is what makes hackathons great : the developer of the library you use is in the same room and you can ask him "Willem, could you add this cool feature ?", and a few minutes after, the feature is available in a git development branch. Basically, the new stuff uses the `return_call_reporting` getdns extension :

```
getdns_dict_set_int(extensions, "return_call_reporting",
                            GETDNS_EXTENSION_TRUE);
```

and then you have a dictionary member `call_reporting` in the answer :

```
getdns_list    *report_list;
getdns_dict    *report_dict;
getdns_dict_get_list(this_response, "call_reporting", &report_list);
getdns_list_get_dict(report_list, 0, &report_dict);
```

The dictionary in the report has now a new member, `tls_peer_cert` (it will appear in getdns 1.1) :

```
getdns_bindata *cert;
getdns_dict_get_bindata(report_dict, "tls_peer_cert", &cert);
```

To parse this certficate (which is in DER format), I use GnuTLS :

```
gnutls_datum_t  raw_cert;
time_t          expiration_time;
struct tm      *f_time;
raw_cert.size = cert->size;
raw_cert.data = malloc(cert->size);
memcpy(raw_cert.data, cert->data, cert->size);
gnutls_x509_crt_import(parsed_cert, &raw_cert, GNUTLS_X509_FMT_DER);
expiration_time = gnutls_x509_crt_get_expiration_time(parsed_cert);
strftime(msgbuf, 1000, "%Y-%m-%d", f_time);
printf("Expires on %s\n", msgbuf);
```

Now, I can test things like an incoming expiration of the certificate.

Another touchy issue was authentication. RFC 7858 allows to authenticate the server by a pinned cryptographic key. (Another authentication methods are under development at the IETF, see draft-ietf-dprive-dtls- That's another problem for Go, by the way : authentication is inflexible, and done by the TLS library. For getdns, on the contrary, is easy : just provide the pinned keys and getdns does the necessary checks :

```
keys = getdns_pubkey_pin_create_from_string(this_context, raw_keys);
getdns_list    *keys_list = getdns_list_create();
getdns_list_set_dict(keys_list, 0, keys);
getdns_dict_set_list(this_resolver, "tls_pubkey_pinset", keys_list);
```

and the result of the authentication is reported in the "call reporting" dictionary we already saw :

```
getdns_bindata *auth_status;
getdns_dict_get_bindata(report_dict, "tls_auth_status", &auth_status);
printf("Authentication is %s\n", auth_status->data);
```

Now, let's put it all together, compile and test from the command line (the arguments are the standard ones for the monitoring plugins, the servers are public servers <https://portal.sinodun.com/wiki/display/TDNS/DNS-over-TLS+test+servers>):

```
% ./check-dns-with-getdns -H 2620:ff:c000:0:1::64:25 -n www.ietf.org
GETDNS OK - 121 ms, expiration date 2027-08-25, auth. Failed:  Address 2400:cb00:2048:1::6814:55 Address 2400:cb

% echo $?
0
```

(We ask the return code of the command but this is what the monitoring software uses to find out whether everything is fine or not.) The authentication status was "Failed" because the server uses a self-signed certificate (otherwise, we would have obtained "None"). Here, we did not require authentication, so the global result is still OK. Should we provide the pinned key, it would be better :

```
% ./check-dns-with-getdns -H 2620:ff:c000:0:1::64:25 -n www.afnic.fr -k pin-sha256=\"pOXrpUt9kgPgbWxBFFcBTbRH2he
GETDNS OK - 1667 ms, expiration date 2027-08-25, auth. Success:  Address 2001:67c:2218:30::24 Address 192.134.5.

% echo $?
0
```

If the key is wrong, it fails :

```
% ./check-dns-with-getdns -H 2620:ff:c000:0:1::64:25 -n www.afnic.fr -a -k pin-sha256=\"pOXrpUt9kgPgbWxBFFcBTbRI
GETDNS CRITICAL - 123 ms, expiration date 2027-08-25, auth. Failed:  Address 2001:67c:2218:30::24 Address 192.13
```

And if the key is wrong **and** we require authentication (-r), we get a fatal error :

```
 % ./check-dns-with-getdns -H 2620:ff:c000:0:1::64:25 -n www.afnic.fr -r -k pin-sha256=\"pOXrpUt9kgPgbWxBFFc
GETDNS CRITICAL - Error Generic error (1) when resolving www.afnic.fr at 2620:ff:c000:0:1::64:25

% echo $?
2
```

And of course, if the server has no DNS-over-TLS or if the server is down, or access to port 853 blocked, we also get an error :

```
% ./check-dns-with-getdns -H 8.8.8.8 -n www.afnic.fr
GETDNS CRITICAL - Error Generic error (1) when resolving www.afnic.fr at 8.8.8.8

% echo $?
2
```

(You can also appreciate the lack of details in error messages...)

By the way, how do you find the key of an existing server? Simplest way is with the `gnutls-cli` program, shipped with GnuTLS :

```
%  gnutls-cli -p 853 2001:4b98:dc2:43:216:3eff:fea9:41a
...
pin-sha256:pAhhaG82hLpW/qXpEIuCknyIYM5iWnbMKsX12rSGm54=
```

(Yes, I know, it works only when you are **not** using Diffie-Hellman.)

Once it is tested, we can put it in a monitoring program. I choosed Icinga. The configuration is :

```
object CheckCommand "dns_with_getdns" {
  command = [ PluginContribDir + "/check_dns_with_getdns" ]

  arguments = {
        "-H" = "$address$",
        "-n" = "$dns_with_getdns_lookup$",
        "-a" = "$dns_with_getdns_authenticate$",
        "-e" = "$dns_with_getdns_accept_errors$",
        "-r" = "$dns_with_getdns_require_auth$",
"-k" = "$dns_with_getdns_keys$",
        "-C" = "$dns_with_getdns_certificate$"
        }
}

apply Service "dns-tls" {
  import "generic-service"

  check_command = "dns_with_getdns"
  assign where (host.address || host.address6) && host.vars.dns_over_tls
  vars.dns_with_getdns_lookup = "www.ietf.org"
  vars.dns_with_getdns_certificate = "7,3"
  vars.dns_with_getdns_accept_errors = false
}

object Host "oarc-dns" {
  import "generic-host"

  address = "2620:ff:c000:0:1::64:25"

  vars.dns_over_tls = true

  vars.dns_with_getdns_authenticate = true
  vars.dns_with_getdns_keys = "pin-sha256=\"pOXrpUt9kgPgbWxBFFcBTbRH2heo2wHwXp1fd4AEVXI=\""
}
```

Then we get the goal of every hackathon project : a screenshot .

Later, this code was used in the nice Project dnsprivacy-monitoring <https://dnsprivacy.org/jenkins/job/dnsprivacy-monitoring/>, which monitors all the public DNS-over-TLS resolvers.

Now, I'm not sure if I'll have time to continue to work on this project. There are several TODO in the code, and an ambitious goal : turn it into a proper plugin suitable for inclusion on the official Monitoring Plugins project. Even better would be to have a generic DNS checker based on getdns, replacing the existing plugins which depend on external commands such as dig. If you want to work on it, the code is at Github <https://github.com/bortzmeyer/monitor-dns-over-tls>.

Many thanks to Willem Toorop for a lot of help and getdns additions, to Francis Dupont for debugging a stupid C problem with GnuTLS (garbage data, unaligned access, all the pleasures of C programming), and to Sara Dickinson for help, inspiration and animation of the DNS team.