Using a TLS proxy

How to use a TLS proxy with a DNS nameserver

It is of course possible to configure a TLS proxy in front of a DNS nameserver to provide DNS-over-TLS. Example configurations for nginx and haproxy are given here.

A more comprehensive setup guide using Docker has been provided by Warren Kumari: dprive-nginx-bind (Thanks Warren!)

Limitations

One of the limitations of using a proxy is that without additional work this will normally mean that the client address is not visible to the nameserver, which can cause issues with NAT64, RRL, RPZ etc. There are ways round this using address re-write rules but we don't detail that here. In future a solution like https://datatracker.ietf.org/doc/draft-bellis-dnsop-xpf/ might become standard.

Nameserver config

To use the following with BIND to offer a TLS service, configure BIND based on the following named.conf snippet

- This assumes BIND 9.12 which supports response padding (comment that line out if you are using an earlier version)
- The rncd and logging config is used to capture traffic volume stats - statistics can be dumped periodically with the 'rndc stats' command

```
options {
        directory "/home/sinodun";
        listen-on port 9999 { 127.0.0.1; };
        allow-query { 127.0.0.1; };
        tcp-clients 4000;
    statistics-file "/tmp/bind-stats";
        dnssec-enable yes;
        dnssec-validation auto;
        response-padding { any; } block-size 468;
};

# Use with the following in named.conf, adjusting the allow list as needed:
key "rndc-key" {
         algorithm hmac-md5;
         secret "BIGSECRET";
};

controls {
         inet 127.0.0.1 port 9953
                 allow { 127.0.0.1; } keys { "rndc-key"; };
};

logging {
        category default { null; };
        category unmatched { null; };
};
```

nginx.conf

```
user www-data;
worker_processes auto;
pid /run/nginx.pid;

events {
worker_connections 1024;
# multi_accept on;
}

stream {
upstream dns_tcp_servers {
server 127.0.0.1:9999;

}

server {
listen 853 ssl;
proxy_pass dns_tcp_servers;

   ssl_certificate       /etc/nginx/lego/certificates/<cert>.crt;
   ssl_certificate_key   /etc/nginx/lego/certificates/<cert>.key;
   ssl_protocols         TLSv1.2;
   ssl_ciphers           ECDHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES128-GCM-SHA256:DHE-
RSA-AES256-GCM-SHA384;
   ssl_session_tickets on;
   ssl_session_timeout   4h;
   ssl_handshake_timeout 30s;
}
}
```

haproxy.cfg

```
global
log /dev/log local0
chroot /var/lib/haproxy
user haproxy
group haproxy
maxconn 1024
pidfile /var/run/haproxy.pid
nbproc <processes>
tune.ssl.default-dh-param 2048
ssl-default-bind-ciphers ECDHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES128-GCM-SHA256:DHE-
RSA-AES256-GCM-SHA384
ssl-default-bind-options force-tlsv12

   # Default SSL material locations
   ca-base /etc/ssl/certs
   crt-base /etc/ssl/private
defaults
balance roundrobin
timeout http-request 10s
timeout queue 1m
timeout connect 10s
timeout client 1m
timeout server 1m
timeout check 10s

listen dns
bind 145.100.185.15:853 ssl crt /etc/haproxy/lego/certificates/<cert>.pem
mode tcp
server server1 127.0.0.1:9999
```

For all but lightly loaded systems, you will need to tune the number of processes or threads available to HAProxy. Unlike the nginx configuration above, which specifies an automatic configuration of the number of worker processes, HAProxy needs to have these quantities set by hand. The simplest way is to set configuration item `nbproc` to an appropriate number; we suggest the number of threads or processes used by the nameserver. For more advanced tuning options, including setting CPU affinity, see the HAProxy documentation or this blog post.

If you use HAProxy and have generated your certificates from Let's Encrypt then you need to combine the certificate chain and key into one file using a command similar to:

```
cat  /etc/letsencrypt/certs/000<N>_chain.pem /etc/letsencrypt/keys/<my_key>.key > /etc/letsencrypt/certs/<cert>.
pem
```