Geoff Huston
November 2017

Three DNS articles:

2. Hiding the DNS

Among all the working groups that met at IETF 100 in Singapore was the first meeting of the DNS over HTTPs Working Group (DOH). I wrote on a related topic of DNS Privacy a little over a year ago (http://bit.ly/2jFLEIO), looking at the work at the time on the privacy-related topics of QNAME minimisation, DNS over TLS and what I called then "Secure DNS over JSON", which was a variant of DNS queries over HTTPS. In this article, I'd like to look in a little more detail at the efforts to hide the DNS behind HTTPS, and put the work in the DOH Working Group into a broader perspective.

There are a number of possible approaches here, and they can be classified according to the level of interaction between the DNS application and the underlying HTTPS encrypted session.

The HTTPS PROXY Tunnel

The most basic concept is to use the HTTPS session as a secure tunnel, where IP packets are tunnelled in HTTP requests in both directions. The client passes data to the server via a client issuing HTTP PUT requests and the reverse path is supported via server performing a GET request. An example of this approach is available in a github repository (http://bit.ly/2zjKREp).

In this case, the HTTP protocol exchange has no awareness of the content that is being passed within the tunnel, and this HTTPS substrate simply functions as an encapsulation medium for the binary DNS queries and responses.

The result is a capability to connect to a remote DNS server while ensuring that the DNS queries and responses are opaque in the client's network. This approach requires some preparation, as the client on the "inside" part of the HTTPS tunnel needs to use a cooperating "outside" rendezvous point. This is similar in many ways to the approach of SSL tunnels (although these days maybe they should be termed "TLS tunnels") where the point-to-point encrypted channel is treated as a virtual interface by the hosts, and IP packets are passed across the virtual circuit.

The HTTPS CONNECT Method

The "CONNECT" method in HTTPS wraps up the functionality in the above Proxy Tunnel model and uses an HTTP command to achieve much the same result, although it is limited to TCP connections, rather than raw IP. A client issues the HTTP CONNECT command to a HTTPS proxy. The proxy then makes a TCP connection to a particular server and port, and relays data between the server's tunnel interface and the target service. It should come as no surprise that servers that support the CONNECT function normally do so in tightly controlled ways.

This approach can be combined with DNS over TCP to define a local DNS resolver proxy that establishes a TCP session with a remote DNS resolver and then passes local queries through the tunnel interface, returning responses recieved from the remote recursive resolver.

DNS via HTTPS

The next approach is to associate the HTTPS server directly with DNS resolution. In this case, the server will look at the parameters associated with the connection, and interpret those parameters as the arguments of a conventional DNS query.

If the server is a recursive DNS resolver, then this allows a remote client to pass a resolution query to the resolver via HTTPS, and receive the response as an HTTP response. In this case, we are stepping away from the DNS resolution protocol as a binary protocol and, instead, we are using ascii text for both the query and the response. The query, and the arguments to the query can be readily expressed as arguments to an HTTP request, using URI-styled semantics. The output however is a problem here. Finding a suitable output text format was a challenge until JSON came along.

The server at https://dns.google.com performs a resolution function over TLS using port 443 with the results passed back as a JSON data structure. This can readily be transformed into an alternative form of gethostbyname() by the application substituting a web object retrieval for a conventional DNS query. This offers the caller some level of privacy from third party inspection and potential intrusion and censorship, although it's unclear precisely what "privacy" means when you are sharing your DNS activity with Google!

Example script:

```
#! /usr/bin/env python
import json, requests

url = "https://dns.google.com/resolve"
params = dict(
    name='www.potaroo.net',
    type='A',
    dnssec='true'
)

resp = requests.get(url=url, params=params)
data = json.loads(resp.text)
print data[u'Answer'][0][u'data']
```

This allows DNS names to be referenced as URIs. For example, the URI https://dns.google.com/resolve?name=www.potaroo.net&type=AAAA specifies a DNS resolution of the name www.potaroo.net for a IPv6 address record, using the Google public DNS server as the recursive resolver.

DNS queries via a URI

There have been efforts in the past to fold the encoding of DNS queries into a dedicated URI scheme. RFC4501 describes such a URI scheme using the string `dns` as the URI scheme identifier . For example, `dns:www.potaroo.net` and `dns://8.8.8.8/www.potaroo.net?type=AAAA` are instances of this form of URI encoding, the first describing DNS resolution of the DNS name `www.potaroo.net` using the browser's default DNS resolver to perform recursive resolution of the DNS query, while the second directs the query to the server addressed by the IP address 8.8.8.8 and requests resolution of the name to an IPv6 address.

This URI does not in and of itself generate a DNS query, but when the URI is passed to an HTTP server, a browser that supports this URI scheme would perform a DNS lookup, and respond with a 'document' that presumably is a binary response that a resolver would conventionally receive from an analogous DNS query.

This approach was documented in 2006, and in the intervening 11 years has not gained much traction, as most browsers do not appear to support this URI scheme, so I guess it's largely defunct these days

DOH: DNS over HTTPS

We are now in a position to look at the standardisation work in the IETF where the DOH Working Group is attempting to standardise DNS over HTTPs. To quote from the Working Group's Charter:

> This working group will standardize encodings for DNS queries and responses that are suitable for use in HTTPS. This will enable the domain name system to function over certain paths where existing DNS methods (UDP, TLS [RFC 7857], and DTLS [RFC 8094]) experience problems.
>
> The working group will re-use HTTPS methods, error codes, and other semantics to the greatest extent possible. The use of HTTPS and its existing PKI provides integrity and confidentiality, and it also allows interoperation with common HTTPS infrastructure and policy.
>
> The primary focus of this working group is to develop a mechanism that provides confidentiality and connectivity between DNS clients (e.g., operating system stub resolvers) and recursive resolvers.

What approach are they using here?

New URI labels for different URI name schemes are unfashionable these days, and the DOH work is no exception. Rather than recycle the `dns:` prefix, the approach uses the currently preferred mechanism, namely the use of a well known URI path of "`.well-known/dns-query`". In this scheme a DNS query would look like:

```
:method = POST
:scheme = https
:authority = dnsserver.example.net
:path = /.well-known/dns-query
accept = application/dns-udpwireformat
content-type = application/dns-udpwireformat
content-length = 33

<33 bytes represented by the following hex encoding>
abcd 0100 0001 0000 0000 0000 0377 7777
0765 7861 6d70 6c65 0363 6f6d 0000 0100
01
```

This approach uses the existing binary DNS on-the-wire format, using a MIME data type of `application/dns-udpwireformat`. When using the `GET` method the DNS query is encoded in Base64, while the POST method places the binary DNS query in the body of the POSTed HTTP object, as per the above example.

I must admit that I am struggling to understand where the win is here. The additional HTTP wrapping seems to add little other than extraneous window dressing, and apart from the cheap thrills in trying to wrap one protocol in the guise of another I can't see the advantage of this approach! If what we are after is the simple ability to conceal DNS queries within an encrypted channel then TLS is doing all the heavy lifting here, and the HTTP component appears to be a source of complication. So why not just use DNS over TLS?

DNS over TLS

The story broke in late October 2017 that several recent commits of the Android code mean that Android will soon support DNS over TLS, evidently intended to enhance the privacy of users when using the DNS (http://bit.ly/2mGyeNu). Considering that Android is used by the overwhelming majority of mobile platforms, and mobile platforms dominate the user-powered Internet, allowing users to hide

their DNS conversations between the handset and their chosen DNS recursive resolver has major implications in the visibility of the DNS.

TLS lies at the heart of HTTPS and is a distinct functional module, operating as a layer above a conventional TCP session. Once a TCP session is established, the Client offers the server its TLS version and enumerates its capabilities. The server chooses from the client-offered capabilities and sends its server certificate. The client generates a PreMasterKey, and encrypts it using the server's public key. The server uses its private key to decrypt this message, and the session key is then established.

The DNS was envisaged as a transaction protocol, and UDP was well suited to this role. TLS is a TCP session protocol and as such it is not universally applicable to all parts of the DNS. However, there is one component of the DNS where a session protocol can make some sense, namely the communication between a client and the client's chosen recursive resolver. Because a client will presumably make repeated calls to the same DNS resolver, then putting this sequence of DNS transactions into a long-lived TCP session is not completely silly! Yes, it's less efficient than UDP, but the TCP session overhead is defrayed across the lifetime of the client / resolver relationship. This is where DNS over TLS can operate efficiently.

Externally, there is little difference between DNS over TLS and DNS over HTTPS. If they both chose to use TCP port 443 then from the outside of the encrypted tunnel there is little to distinguish the two. Both are encrypted TCP sessions, and if you were careful with the use of padding, you could probably make them completely indistinguishable.

However, in both cases the destination service name is visible in the Server Name Indication part of the unencrypted part of the TLS exchange. If you wanted to further obscure this part of the TLS session you need to encrypt the SNI part of the TLS exchange as well. This is still under development in the IETF, and one approach that uses TLS tunnelling is visible in an internet draft (https://bit.ly/2hSyAeL).

DTLS

Last and by no means least there is TLS over a datagram transport, DTLS. Described for TLS 1.2 in RFC6347. There are some issues here, not the least is DTLS' insensitivity to UDP packet fragmentation. If it is necessary to end a DTLS packet larger than the Path MTU the server should indicate its inability to do so using the DNS truncated flag, and signal that the client should re-query using DNS over TLS.

The specification of DNS over DTLS, RFC8094, contains an interesting caveat:

> This DTLS solution was considered by the DPRIVE working group as an option to use in case the TLS-based approach specified in [RFC7858] turns out to have some issues when deployed. At the time of writing, it is expected that [RFC7858] is what will be deployed, and so this specification is mainly intended as a backup.

It seems reasonable to conclude that there is not much interest in DNS over DTLS at this point in time, and the RFC was published more as a formality than a statement of intent!

What to Use?

I think the Android developers got it right when they opted to incorporate DNS over TLS in their platform.

The use of TLS has some server overhead in terms of session state, but at the same time in the area of client to recursive resolver these session overheads can be defrayed over a long session time. This approach avoid the Path MTU considerations and fallback to DNS over TCP for large responses that is required by DNS over DTLS. This approach also avoids what I can only interpret as an unnecessary HTTP session layer intrusion with DNS over HTTPS.

Personally, I'm keen to get hold of an implementation of DNS over TLS and see how it performs!

Author

Geoff Huston B.Sc., M.Sc., is the Chief Scientist at APNIC, the Regional Internet Registry serving the Asia Pacific region.

www.potaroo.net

Disclaimer

The above views do not necessarily represent the views or positions of the Asia Pacific Network Information Centre.