DNS Queries over HTTPS
draft-ietf-doh-dns-over-https-02

Abstract

DNS queries sometimes experience problems with end to end connectivity at times and places where HTTPS flows freely.

HTTPS provides the most practical mechanism for reliable end to end communication. Its use of TLS provides integrity and confidentiality guarantees and its use of HTTP allows it to interoperate with proxies, firewalls, and authentication systems where required for transit.

This document describes how to run DNS service over HTTP using https:// URIs.

[[ There is a repository for this draft at https://github.com/dohwg/draft-ietf-doh-dns-over-https.]]

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on June 1, 2018.
1. Introduction

The Internet does not always provide end to end reachability for native DNS. On-path network devices may spoof DNS responses, block DNS requests, or just redirect DNS queries to different DNS servers that give less-than-honest answers.
Over time, there have been many proposals for using HTTP and HTTPS as a substrate for DNS queries and responses. To date, none of those proposals have made it beyond early discussion, partially due to disagreement about what the appropriate formatting should be and partially because they did not follow HTTP best practices.


The described approach is more than a tunnel over HTTP. It establishes default media formatting types for requests and responses but uses normal HTTP content negotiation mechanisms for selecting alternatives that endpoints may prefer in anticipation of serving new use cases. In addition to this media type negotiation, it aligns itself with HTTP features such as caching, proxying, and compression.

The integration with HTTP provides a transport suitable for both traditional DNS clients and native web applications seeking access to the DNS.

2. Terminology

A server that supports this protocol is called a "DNS API server" to differentiate it from a "DNS server" (one that uses the regular DNS protocol). Similarly, a client that supports this protocol is called a "DNS API client".

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in BCP 14, RFC 2119 [RFC2119].

3. Use Cases

There are two initial use cases for this protocol.

The primary use case is to prevent on-path network devices from interfering with DNS operations. This interference includes, but is not limited to, spoofing DNS responses, blocking DNS requests, and tracking.

In this use, clients - whether operating systems or individual applications - will be explicitly configured to use a DOH server as a recursive resolver by its user (or administrator). They might use
the DOH server for all queries, or only for a subset of them. The specific configuration mechanism is out of scope for this document.

A secondary use case is allowing web applications to access DNS information, by using existing APIs in browsers to access it over HTTP in a safe way consistent with Cross Origin Resource Sharing (CORS) [CORS].

This is technically already possible (since the server controls both the HTTP resources it exposes and the use of browser APIs by its content), but standardisation might make this easier to accomplish.

Note that in this second use, the browser does not consult the DOH server or use its responses for any DNS lookups outside the scope of the application using them; i.e., there is (currently) no API that allows a Web site to poison DNS for others.

[[ This paragraph is to be removed when this document is published as an RFC ]] Note that these use cases are different than those in a similar protocol described at [I-D.ietf-dnsop-dns-wireformat-http]. The use case for that protocol is proxying DNS queries over HTTP instead of over DNS itself. The use cases in this document all involve query origination instead of proxying.

4. Protocol Requirements

The protocol described here bases its design on the following protocol requirements:

- The protocol must use normal HTTP semantics.

- The queries and responses must be able to be flexible enough to express every normal DNS query.

- The protocol must allow implementations to use HTTP's content negotiation mechanism.

- The protocol must ensure interoperable media formats through a mandatory to implement format wherein a query must be able to contain one or more EDNS extensions, including those not yet defined.

- The protocol must use a secure transport that meets the requirements for modern HTTPS.
4.1. Non-requirements

- Supporting network-specific DNS64 [RFC6147]
- Supporting other network-specific inferences from plaintext DNS queries
- Supporting insecure HTTP
- Supporting legacy HTTP

5. The HTTP Request

To make a DNS API query, a DNS API client sends an HTTP request to the URI of the DNS API.

The URI scheme MUST be https.

A client can be configured with a DNS API URI, or it can discover the URI. This document defines a well-known URI path of "/.well-known/dns-query" so that a discovery process that produces a domain name or domain name and port can be used to construct the DNS API URI. (See Section 8 for the registration of this in the well-known URI registry.) DNS API servers SHOULD use this well-known path to help contextualize DNS Query requests that use server push [RFC7540].

A DNS API Client encodes a single DNS query into the HTTP request using either the HTTP GET or POST methods.

When using the POST method the DNS query is included as the message body of the HTTP request and the Content-Type request header indicates the media type of the message. POST-ed requests are smaller than their GET equivalents.

When using the GET method the URI path MUST contain a query parameter with the name of ct and a value indicating the media-format used for the body parameter. The value may either be an explicit media type (e.g. ct=application/dns-udpwireformat&body=...) or it may be empty. An empty value indicates the default application/dns-udpwireformat type (e.g. ct&body=...).

When using the GET method the URI path MUST contain a query parameter with the name of body. The value of the parameter is the content of the request encoded with base64url [RFC4648]. Using the GET method is friendlier to many HTTP cache implementations.

The DNS API Client SHOULD include an HTTP "Accept:" request header to say what type of content can be understood in response. The client
MUST be prepared to process "application/dns-udpwireformat" responses but MAY process any other type it receives.

In order to maximize cache friendliness, DNS API clients using media formats that include DNS ID, such as application/dns-udpwireformat, SHOULD use a DNS ID of 0 in every DNS request. HTTP correlates request and response, thus eliminating the need for the ID in a media type such as application/dns-udpwireformat and the use of a varying DNS ID can cause semantically equivalent DNS queries to be cached separately.

DNS API clients can use HTTP/2 padding and compression in the same way that other HTTP/2 clients use (or don't use) them.

5.1. DNS Wire Format

The media type is "application/dns-udpwireformat". The body is the DNS on-the-wire format is defined in [RFC1035].

When using the GET method, the body MUST be encoded with base64url [RFC4648]. Padding characters for base64url MUST NOT be included.

When using the POST method, the body is not encoded.

DNS API clients using the DNS wire format MAY have one or more EDNS(0) extensions [RFC6891] in the request.

5.2. Examples

These examples use HTTP/2 style formatting from [RFC7540].

For this example assume a DNS API server is following this specification on origin https://dnsserver.example.net/ and the well-known path. The DNS API client chooses to send its requests in application/dns-udpwireformat but indicates it can parse replies in that format or as a hypothetical JSON-based content type. The application/simpledns+json type used by this example is currently fictitious.

```
:method = GET
:scheme = https
:authority = dnsserver.example.net
:path = /.well-known/dns-query?ct&  (no CR)
      body=q88BAABAAABAAAAAAA3d3dwdleGFtcGxlA2NvbQAAAQAB
accept = application/dns-udpwireformat, application/simpledns+json
```

The same DNS query, using the POST method would be:
6. The HTTP Response

Different response media types will provide more or less information from a DNS response. For example, one response type might include the information from the DNS header bytes while another might omit it. The amount and type of information that a media type gives is solely up to the format, and not defined in this protocol.

At the time this is published, the response types are works in progress. The only known response type is "application/dns-udpwireformat", but it is possible that at least one JSON-based response format will be defined in the future.

The DNS response for "application/dns-udpwireformat" in Section 5.1 MAY have one or more EDNS(0) extensions, depending on the extension definition of the extensions given in the DNS request.

Native HTTP methods are used to correlate requests and responses. Responses may be returned in a different temporal order than requests were made using the protocols native multi-streaming functionality.

The Answer section of a DNS response contains one or more RRsets. (RRsets are defined in [RFC7719].) According to [RFC2181], each resource record in an RRset is supposed to have the Time To Live (TTL) freshness information. Different RRsets in the Answer section can have different TTLs, though it is only possible for the HTTP response to have a single freshness lifetime. The HTTP response freshness lifetime ([RFC7234] Section 4.2) should be coordinated with the Resource Record bearing the smallest TTL in the Answer section of the response. The HTTP freshness lifetime SHOULD be set to expire at the same time any of the DNS Records reach a 0 TTL. The response freshness lifetime MUST NOT be greater than that indicated by the DNS Record with the smallest TTL in the response.
A DNS API Client that receives a response without an explicit freshness lifetime MUST NOT assign that response a heuristic freshness ([RFC7234] Section 4.2.2.) greater than that indicated by the DNS Record with the smallest TTL in the response.

A DNS API Server MUST be able to process application/dns-udpwireformat request messages.

A DNS API Server SHOULD respond with HTTP status code 415 upon receiving a media type it is unable to process.

This document does not change the definition of any HTTP response codes or otherwise proscribe their use.

HTTP revalidation of cached DNS information may be of limited value as revalidation provides only a bandwidth benefit and DNS transactions are normally latency bound instead. Furthermore, the HTTP response headers that enable revalidation (such as "Last-Modified" and "Etag") are often fairly large when compared to the overall DNS response size, and have a variable nature that creates constant pressure on the HTTP/2 compression dictionary [RFC7541]. Other types of DNS data, such as zone transfers, may be larger and benefit more from revalidation. DNS API servers may wish to consider whether providing these optional response headers is worthwhile.

6.1. Example

This is an example response for a query for the IN A records for "www.example.com" with recursion turned on. The response bears one record with an address of 93.184.216.34 and a TTL of 128 seconds.

:status = 200
color-type = application/dns-udpwireformat
color-length = 64
cache-control = max-age=128

<64 bytes represented by the following hex encoding>
abcd 8180 0001 0001 0000 0000 0377 7777
0765 7861 6d70 6c65 0363 6f6d 0000 0100
0103 7777 7707 6578 616d 706c 6503 636f
6d00 0001 0001 0000 0080 0004 5db8 d822

7. HTTP Integration

This protocol MUST be used with https scheme URI [RFC7230].
7.1. HTTP/2

The minimum version of HTTP used by DOH SHOULD be HTTP/2 [RFC7540]. The messages in classic UDP based DNS [RFC1035] are inherently unordered and have low overhead. A competitive HTTP transport needs to support reordering, parallelism, priority, and header compression to achieve similar performance. Those features were introduced to HTTP in HTTP/2 [RFC7540]. Earlier versions of HTTP are capable of conveying the semantic requirements of DOH but would result in very poor performance for many uses cases.

8. IANA Considerations

8.1. Registration of Well-Known URI

This specification registers a Well-Known URI [RFC5785]:

- URI Suffix: dns-query
- Change Controller: IETF
- Specification Document(s): [this specification]

8.2. Registration of application/dns-udpwireformat Media Type
To: ietf-types@iana.org
Subject: Registration of MIME media type
        application/dns-udpwireformat

MIME media type name: application
dns-udpwireformat

Required parameters: n/a
Optional parameters: n/a

Encoding considerations: This is a binary format. The contents are a
DNS message as defined in RFC 1035. The format used here is for DNS
over UDP, which is the format defined in the diagrams in RFC 1035.

Security considerations: The security considerations for carrying
this data are the same for carrying DNS without encryption.

Interoperability considerations: None.

Published specification: This document.

Applications that use this media type:
Systems that want to exchange full DNS messages.

Additional information:
Magic number(s): n/a
File extension(s): n/a
Macintosh file type code(s): n/a

Person & email address to contact for further information:
Paul Hoffman, paul.hoffman@icann.org

Intended usage: COMMON

Restrictions on usage: n/a

Author: Paul Hoffman, paul.hoffman@icann.org

Change controller: IESG
9. Security Considerations

Running DNS over HTTPS relies on the security of the underlying HTTP transport. Implementations utilizing HTTP/2 benefit from the TLS profile defined in [RFC7540] Section 9.2.

Session level encryption has well known weaknesses with respect to traffic analysis which might be particularly acute when dealing with DNS queries. Sections 10.6 (Compression) and 10.7 (Padding) of [RFC7540] provide some further advice on mitigations within an HTTP/2 context.

The HTTPS connection provides transport security for the interaction between the DNS API server and client, but does not inherently ensure the authenticity of DNS data. A DNS API client may also perform full DNSSEC validation of answers received from a DNS API server or it may choose to trust answers from a particular DNS API server, much as a DNS client might choose to trust answers from its recursive DNS resolver.

[[ From the WG charter:

The working group will analyze the security and privacy issues that could arise from accessing DNS over HTTPS. In particular, the working group will consider the interaction of DNS and HTTP caching.

]]

A server that is acting both as a normal web server and a DNS API server is in a position to choose which DNS names it forces a client to resolve (through its web service) and also be the one to answer those queries (through its DNS API service). An untrusted DNS API server can thus easily cause damage by poisoning a client's cache with names that the DNS API server chooses to poison. A client MUST NOT trust a DNS API server simply because it was discovered, or because the client was told to trust the DNS API server by an untrusted party. Instead, a client MUST only trust DNS API server that is configured as trustworthy.

[[ From the WG charter:

The working group may define mechanisms for discovery of DOH servers similar to existing mechanisms for discovering other DNS servers if the chairs determine that there is both sufficient interest and working group consensus.

]]
10. Operational Considerations

Local policy considerations and similar factors mean different DNS servers may provide different results to the same query: for instance in split DNS configurations [RFC6950]. It logically follows that the server which is queried can influence the end result. Therefore a client's choice of DNS server may affect the responses it gets to its queries.

The HTTPS channel used by this specification establishes secure two party communication between the DNS API Client and the DNS API Server. Filtering or inspection systems that rely on unsecured transport of DNS will not function in a DNS over HTTPS environment.

Many HTTPS implementations perform real time third party checks of the revocation status of the certificates being used by TLS. If this check is done as part of the DNS API server connection procedure and the check itself requires DNS resolution to connect to the third party a deadlock can occur. The use of an OCSP [RFC6960] server is one example of how this can happen. DNS API servers SHOULD utilize OCSP Stapling [RFC6961] to provide the client with certificate revocation information that does not require contacting a third party.

11. Acknowledgments

Joe Hildebrand contributed lots of material for a different iteration of this document. Helpful early comments were given by Ben Schwartz and Mark Nottingham.

12. References

12.1. Normative References


[RFC6950]"
12.2. Informative References

Appendix A. Previous Work on DNS over HTTP or in Other Formats

The following is an incomplete list of earlier work that related to DNS over HTTP/1 or representing DNS data in other formats.

The list includes links to the tools.ietf.org site (because these documents are all expired) and web sites of software.

- [https://www.nlnetlabs.nl/projects/dnssec-trigger/](https://www.nlnetlabs.nl/projects/dnssec-trigger/)
Authors' Addresses

Paul Hoffman
ICANN
Email: paul.hoffman@icann.org

Patrick McManus
Mozilla
Email: pmcmanus@mozilla.com