dprive
Internet-Draft
Intended status: Informational
Expires: October 26, 2017

D. Gillmor
ACLU

April 24, 2017

Demultiplexing Streamed DNS from HTTP
draft-dkg-dprive-demux-dns-http-00

Abstract

DNS over TCP and traditional HTTP are both stream-oriented, client-speaks-first protocols. They can both be run over a stream-based security protocol like TLS. A server accepting a stream-based client can distinguish between a valid stream of DNS queries and valid stream of HTTP requests by simple observation of the first few octets sent by the client. This can be done without any external demultiplexing mechanism like TCP port number or ALPN.

Implicit multiplexing of the two protocols over a single listening port can be useful for obscuring the presence of DNS queries from a network observer, which makes it relevant for DNS privacy.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

DNS and HTTP are both client-speaks-first protocols capable of running over stream-based transport like TCP, or as the payload of a typical TLS [RFC5246] session.

There are some contexts where it is useful for a server to be able to decide what protocol is used by an incoming TCP stream, to choose dynamically between DNS and HTTP on the basis of the stream itself (rather than a port designation or other explicit demultiplexing).

For example, a TLS terminator listening on port 443 might be willing to serve DNS-over-TLS [RFC7858] as well as HTTPS.

A simple demultiplexing server should do this demuxing based on the first few bytes sent by the client on a given stream; once a choice has been established, the rest of the stream is committed to one or the other interpretation.

This document provides proof that a demultiplexer can robustly distinguish HTTP from DNS on the basis of the content of the stream alone.

A DNS client that knows it is talking to a server which is this position (e.g. trying to do DNS-over-TLS on TCP port 443, used traditionally only for HTTPS) might also want to be aware of network traffic patterns that could confuse such a server. This document presents explicit mitigations that such a DNS client MAY decide to use.

This document limits its discussion of HTTP over TCP or TLS or some other classical stream-based protocol (it excludes HTTP over QUIC, for example). Likewise, it considers only the TCP variant of DNS (and excludes DNS over UDP or any other datagram transport).

FIXME: address network stack ossification here?

2. Distinguish only at the start of a stream

A server which attempts to distinguish DNS queries from HTTP requests individually might consider using these guidelines in the middle of a running stream (e.g. at natural boundaries, like the end of an HTTP request, or after a DNS message), but this document focuses specifically on a heuristic choice for the whole stream, based on the initial few octets sent by the client.

While it's tempting to consider distinguishing at multiple points in the stream, the complexities of determining the specific end of an

HTTP/1.1 request body, and the difficulty in distinguishing an HTTP/2 frame header from a streamed DNS message make this more difficult to implement. Interleaving the responses themselves on a stream with multiple data elements is also challenging. So do not use this technique anywhere but at the beginning of a stream!

If being able to interleave DNS queries with HTTP requests on a single stream is desired, a strategy like [I-D.ietf-dnsop-dns-wireformat-http] is recommended instead.

2.1. Why not ALPN?

If this is done over TLS, a natural question is whether the client should simply indicate its preferred protocol in the TLS handshake's ALPN [RFC7301] extension.

However, ALPN headers are visible to a network observer, and a network controller attempting to confine the user's DNS traffic to a limited set of servers could use the ALPN header as a signal to block DNS-specific streams.

3. Overview of initial octets

3.1. DNS stream initial octets

[RFC1035] section 4.2.2 ("TCP Usage") shows that every stream-based DNS connection starts with a DNS message, preceded with a 2-octet message length field:

The message is prefixed with a two byte length field which gives the message length, excluding the two byte length field.

[RFC1035] section 4.1.1 represents the DNS message header section, which is the first part of the DNS message on the wire (after the message length).

```
                                1  1  1  1  1  1
  0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                      ID                       |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|QR|   Opcode  |AA|TC|RD|RA|   Z    |   RCODE   |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    QDCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    ANCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    NSCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    ARCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

So in a DNS over TCP stream, the interpretation of the initial 14 octets are fixed based on information about the first query sent on the stream:

- o 0,1: length of initial DNS message
- o 2,3: DNS Transaction ID
- o 4,5: DNS opcode, flags, and response code
- o 6,7: Question count
- o 8,9: Answer count
- o 10,11: Authority count
- o 12,13: Additional RR count

All DNS streams sent over TCP start with at least these 14 octets.

3.2. HTTP initial octets

In an HTTP stream, the first octets sent from the client are either the so-called "Simple-Request" (for HTTP/0.9), the "Request-Line" (for HTTP/1.0 and HTTP/1.1), which has variable characteristics, or the "connection preface" (for HTTP/2) which is a fixed string.

Some servers may wish to ignore the oldest of these, HTTP/0.9.

3.2.1. HTTP/0.9

[RFC1945] section 4.1 says that HTTP/0.9 queries (that is, HTTP queries from before HTTP/1.0 was formalized) use this form:

```
Simple-Request = "GET" SP Request-URI CRLF
```

Note that HTTP/0.9 clients send this string and only this string, nothing else (no request body, no subsequent requests). The "Request-URI" token is guaranteed to start with a printable ASCII character, and cannot contain any members of the CTL class (values 0x00 through 0x1F) but due to loose early specifications, it might sometimes contain high-valued octets (those with the most-significant bit set - 0x80 or above).

So the first 5 octets are all constrained to be no less than 0x20 (SP) and no more than 0x7F (DEL), and all subsequent octets sent from the client have a value at least 0x0A (LF).

The shortest possible HTTP/0.9 client request is:

```
 char: G  E  T  SP /  CR LF
index: 0  1  2  3  4  5  6
```

The lowest possible HTTP/0.9 client request (sorted ASCIIbetically) is:

```
 char: G  E  T  SP +  :  CR LF
index: 0  1  2  3  4  5  6  7
```

3.2.2. HTTP/1.0 and HTTP/1.1

The request line format for HTTP/1.1 matches that of HTTP/1.0 (HTTP/1.1 adds protocol features like pipelining, but doesn't change the request form itself). But unlike HTTP/0.9, the initial verb (the "method") can vary.

[RFC7230] section 3.1.1 says that the first line of an HTTP/1.1 request is:

```
request-line   = method SP request-target SP HTTP-version CRLF
method         = token
```

and [RFC7230] section 3.2.6 says:

```
 token          = 1*tchar

 tchar          = "!" / "#" / "$" / "%" / "&" / "'" / "*"
                / "+" / "-" / "." / "^" / "_" / "`" / "|" / "~"
                / DIGIT / ALPHA
                ; any VCHAR, except delimiters
```

and VCHAR is defined in [RFC5234] appendix B.1 as:

```
VCHAR          =  %x21-7E
```

"request-target" itself cannot contain 0x20 (SP) or any CTL characters, or any characters above the US-ASCII range (> 0x7F).

And the "HTTP-version" token is either the literal string "HTTP/1.0" or the literal string "HTTP/1.1", both of which are constrained to the same printable-ASCII range.

The ASCIIbetically-lowest shortest possible HTTP/1.0 or HTTP/1.1 request is:

```
 char: !  SP /  H  T  T  P  /  1  .  0  CR LF CR LF
index: 0  1  2  3  4  5  6  7  8  9  0  a  b  c  d
```

In any case, no HTTP/1.0 or HTTP/1.1 request line can include any values lower than 0x0A (LF) or greater than 0x7F (DEL) in the first 14 octets.

However, [RFC7230] section 3.1.1 also says:

In the interest of robustness, a server that is expecting to receive and parse a request-line SHOULD ignore at least one empty line (CRLF) received prior to the request-line.

So we should also consider accepting an arbitrary number of repeated CRLF sequences before the request-line as a potentially-valid HTTP client behavior.

3.2.3. HTTP/2

[RFC7540] section 3.5 says:

In HTTP/2, each endpoint is required to send a connection preface as a final confirmation of the protocol in use and to establish the initial settings for the HTTP/2 connection. The client and server each send a different connection preface.

The client connection preface starts with a sequence of 24 octets, which in hex notation is:

```
0x505249202a20485454502f322e300d0a0d0a534d0d0a0d0a
```

That is, the connection preface starts with the string "PRI * HTTP/2.0\r\n\r\nSM\r\n\r\n").

The highest valued octet here is 0x54 ("T"), and the lowest is 0x0A (LF).

4. Specific octets

The sections below examine likely values of specific octet positions in the stream. All octet indexes are 0-based.

4.1. octets 0 and 1

Any DNS message less than 3338 octets sent as the initial query over TCP can be reliably distinguished from any version of HTTP by the first two octets of the TCP stream alone.

3338 is 0x0D0A, or the ASCII string CRLF, which some HTTP clients might send before an initial request. No HTTP client can legitimately send anything lower than this.

Most DNS queries are easily within this range automatically.

4.2. octets 2 and 3

In a DNS stream, octets 2 and 3 represent the client-chosen message ID. The message ID is used to bind messages with responses. Over connectionless transports like UDP, this is an important anti-spoofing measure, as well as a distinguishing measure for clients reusing the same UDP port for multiple outstanding queries. Standard DNS clients already explicitly randomize this value.

For the connection-oriented streaming DNS discussed here, the anti-spoofing characteristics are not relevant (the connection itself provides anti-spoofing), so the client is free to choose arbitrary values.

With a standard DNS client which fully-randomizes these values, only 25% of generated queries will have the high bits of both octets set to 0. 100% of all HTTP requests will have the high bits of both of these octets cleared. Similarly, some small percentage of randomly-generated DNS queries will have values here lower than 0x0A, while no HTTP clients will ever send these low values.

4.3. octet 4

In a DNS stream, octet 4 combines several fields:

```
  0  1  2  3  4  5  6  7
+--+--+--+--+--+--+--+--+
|QR|   Opcode  |AA|TC|RD|
+--+--+--+--+--+--+--+--+
```

In a standard DNS query sent over a streaming interface, QR, Opcode, AA, and TC are all set to 0. The least-significant bit (RD - Recursion Desired) is set when a packet is sent from a stub to a recursive resolver. The value of such an octet is 0x01. This value never occurs in octet 4 of a legitimate HTTP client.

4.4. octet 5

In a DNS stream, octet 5 also combines several fields:

```
  0  1  2  3  4  5  6  7
+--+--+--+--+--+--+--+--+
|RA|   Z    |   RCODE   |
+--+--+--+--+--+--+--+--+
```

In a standard DNS message sent from a client, all these bits are 0. No legitimate HTTP client ever sends a value of 0x00 in octet 5.

4.5. octets 6 and 7

In DNS, octets 6 and 7 represent the query count. Most DNS clients will send one query at a time, which makes this value 0x0001. As long as the number of initial queries does not exceed 0x0A0A (2570), then at least one of these octets will have a value less than 0x0A. No HTTP client sends an octet less than 0x0A in positions 6 or 7.

4.6. octets 8 through 11

In streaming DNS, octets 8 through 11 represent answer counts and authority counts. Standard DNS queries will set them both 0. No HTTP client sends a zero-valued octet in these positions.

4.7. octets 12 and 13

In streaming DNS, octets 12 and 13 represent the number of Additional RRs. When a DNS query is sent with EDNS(0), the OPT RR is accounted for here. So this is often either 0x0000 or 0x0001. No HTTP client will send these values at these positions.

5. Combinations of octets

In a DNS message, each Question in the Question section is at least 5 octets (1 octet for zero-length QNAME + 2 octets for QTYPE + 2 octets for QCLASS), and each RR (in the Answer, Authority, and Additional sections) is at least 11 octets. And the header itself is 12 octets.

So we know that for a valid DNS stream, the first message has a size of at least:

```
min_first_msg_size = 12 + 5 * (256*o[6] + o[7]) +
                     11 * (256*(o[8] + o[10] + o[12]) +
                                o[9] + o[11] + o[13])
```

It's possible to compare this value with the expected first query size:

```
first_msg_size = 256 * o[0] + o[1]
```

if "first_query_size" is less than "min_first_query_size" we can be confident that the stream is not DNS.

5.1. Proof: a valid DNS message cannot be an HTTP query

For any a valid, stream-based DNS message:

- o If there are fewer than 0x0A00 Questions then octet 6 < 0x0A.
- o If there are fewer than 0x0A00 Answer RRs, then octet 8 < 0x0A.
- o If there are fewer than 0x0A00 Authority RRs, then octet 10 < 0x0A.
- o If there are fewer than 0x0A00 Additional RRs, then octet 12 < 0x0A.

If any of these four inequalities hold, then the packet is clearly DNS, not HTTP.

if none of them hold, then there are at least 0x0A00 (2560) Questions and 3*2560 == 7680 RRs. But:

```
12 + 5*2560 + 11*7680 == 97292
```

So the smallest possible DNS message where none of these four inequalites hold is 97292 octets. But a DNS message is limited in size to 65535 octets.

Therefore at least one of these inequalities holds, and one of the first 14 octets of a DNS steam is < 0x0A.

But in a standard HTTP request, none of the first 14 octets can have a value < 0x0A, so a valid DNS message cannot be mistaken for an HTTP request.

6. Guidance for Demultiplexing Servers

Upon receiving a connection stream that might be either DNS or HTTP, a server can inspect the initial octets of the stream to decide where to send it.

6.1. Without supporting HTTP/0.9

A server that doesn't care about HTTP/0.9 can simply wait for the first 14 octets of the client's request to come in. Then the algorithm is:

```
bytestream = read_from_client(14)
for x in bytestream:
    if (x < 0x0A) or (x > 0x7F):
       return `DNS`
return `HTTP`
```

6.2. Supporting archaic HTTP/0.9 clients

A server that decides to try to support HTTP/0.9 clients has a slightly more challenging task, since some of them may send fewer octets than the initial DNS message, and the server shouldn't block waiting for data that will never come.

```
bytestream = read_from_client(5)
for x in bytestream[0:5]
    if (x < 0x0A) or (x > 0x7F):
       return `DNS`
if (bytestream[0:4] != 'GET '):    # not HTTP/0.9
    bytestream += read_from_client(9)
    for x in bytestream[5:14]:
        if (x < 0x0A) or (x > 0x7f):
            return `DNS`
    return `HTTP`
else:                              # maybe HTTP/0.9
    seen_sp = False
    seen_high = False
    while (len(bytestream) < 14):
        if (seen_sp and seen_high):
            return `DNS`
        x = read_from_client(1)
        bytestream += x
        if (x > 0x7F):
            seen_high = True
        elif (x < 0x0A):
            return `DNS`
        elif (x == 0x20):
            seen_sp = True       # SP found before CRLF, not HTTP/0.9
        elif (x == 0x0A):
            return `HTTP`
    return `HTTP`
```

Note that if read_from_client() ever fails to read the number of requested bytes (e.g. because of EOF), then the stream is neither valid HTTP nor valid DNS, and can be discarded.

6.3. Signaling demultiplexing capacity

FIXME: should there be a way for a listener to signal somehow that it is willing and capable of handling both DNS and HTTP traffic? There would need to be a different signaling mechanism for each stream (unless the signalling is done somehow in an outer layer like TLS). This is probably out-of-scope for this draft.

7. Guidance for DNS clients

Consider a DNS client that connects to a server that might be interested in answering HTTP requests on the same address/port (or other channel identifier). The client wants to send traffic that is unambiguously DNS traffic to make it easy for the server to distinguish it from inbound HTTP requests. Fortunately, this is trivial to do.

Such a client should follow these guidelines:

- o Send the DNS message size (a 16-bit integer) together in the same packet with the full header of the first DNS message so that the recipient can review as much as possible of the frame at once. This is a best practice for efficient stream-based DNS anyway.

If the client is concerned about stream fragmentation that it cannot control, and it is talking to a server that might be expecting HTTP/0.9 clients, then the server might not be willing to wait for the full initial 14 octets to make a decision.

Note that this fragmentation is not a concern for streams wrapped in TLS when using modern AEAD ciphersuites. In this case, the client gets to choose the size of the plaintext record, which is either recovered by the server in full (unfragmented) or the connection fails.

If the client does not have such a guarantee from the transport, it MAY also take one of the following mitigating actions relating to the first DNS message it sends in the stream [explanation of what the server gets to see in the fragmented stream case are in square brackets after each mitigation]:

- o Ensure the first message is marked as a query (QR = 0), and it uses opcode 0 ("Standard Query"). [bytestream[4] < 0x08]

- o Ensure that the first message has RA = 0, Z = 0, and RCODE = 0. [bytestream[5] == 0x00]

- o Ensure that the high bit of the first octet of the message ID of the first message is set. [bytesteam[2] > 0x7F]

- o Send an initial short Server Status DNS message ahead of the otherwise intended initial DNS message. [bytstream[0] == 0x00]

- o Use the EDNS(0) padding option [RFC7830] to pad the first message to a multiple of 256 octets. [bytestream[1] == 0x00]

7.1. Interpreting failure

FIXME: A DNS client that does not already know that a server is willing to carry both types of traffic SHOULD expect a transport connection failure of some sort. Can we say something specific about what it should expect?

8. Guidance for HTTP clients

HTTP clients SHOULD NOT send HTTP/0.9 requests, since modern HTTP servers are not required to support HTTP/0.9. Sending an HTTP/1.0 request (or any later version) is sufficient for a server to be able to distinguish the two protocols.

9. Security Considerations

FIXME: Clients should locally validate DNSSEC (servers may still be able to omit some records)

FIXME: if widely deployed, consider amplification for DDoS against authoritative servers?

FIXME: consider dnssec transparency

FIXME: consider TLS session resumption - this counts as a new stream boundary, so the multiplexing decision need not persist across resumption.

FIXME: consider 0-RTT

FIXME: consider X.509 cert validation

FIXME: what other security considerations should clients take?

FIXME: what other security considerations should servers take?

10. Privacy Considerations

FIXME: DNS queries and HTTP requests can reveal potentially sensitive information about the sender.

FIXME: consider DNS and HTTP traffic analysis - how should requests or responses be padded, aggregated, or delayed given that streams are multiplexed?

FIXME: any other privacy considerations?

11. IANA Considerations

This document does not ask IANA to make any changes to existing registries.

12. Document Considerations

[RFC Editor: please remove this section before publication]

This document is currently edited as markdown. Minor editorial changes can be suggested via merge requests at https://gitlab.com/dkg/hddemux or by e-mail to the author. Please direct all significant commentary to the public IETF DNS Privacy mailing list: dns-privacy@ietf.org

13. References

13.1. Normative References

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <http://www.rfc-editor.org/info/rfc1035>.

[RFC1945] Berners-Lee, T., Fielding, R., and H. Frystyk, "Hypertext Transfer Protocol -- HTTP/1.0", RFC 1945, DOI 10.17487/RFC1945, May 1996, <http://www.rfc-editor.org/info/rfc1945>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <http://www.rfc-editor.org/info/rfc5234>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <http://www.rfc-editor.org/info/rfc7540>.

13.2. Informative References

[I-D.ietf-dnsop-dns-wireformat-http] Song, L., Vixie, P., Kerr, S., and R. Wan, "DNS wire-format over HTTP", draft-ietf-dnsop-dns-wireformat-http-01 (work in progress), March 2017.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

[RFC7301] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <http://www.rfc-editor.org/info/rfc7301>.

[RFC7830] Mayrhofer, A., "The EDNS(0) Padding Option", RFC 7830, DOI 10.17487/RFC7830, May 2016, <http://www.rfc-editor.org/info/rfc7830>.

[RFC7858] Hu, Z., Zhu, L., Heidemann, J., Mankin, A., Wessels, D., and P. Hoffman, "Specification for DNS over Transport Layer Security (TLS)", RFC 7858, DOI 10.17487/RFC7858, May 2016, <http://www.rfc-editor.org/info/rfc7858>.

Author's Address

Daniel Kahn Gillmor
American Civil Liberties Union
125 Broad St.
New York, NY 10004
USA

Email: dkg@fifthhorseman.net