dnsop
Internet-Draft
Intended status: Standards Track
Expires: April 18, 2015

B. Dickson

October 15, 2014

System to transport DNS over HTTP using JSON
draft-dickson-dnsop-spartacus-system-00

Abstract

This is the SPARTACUS DNS gateway system. It is designed to facilitate the transport of DNS messages opaquely, across problematic sections of the Internet. It uses JSON encoding, and HTTP(S) as the protocol for transport.

The main criteria of SPARTACUS is that it preserve DNS messages verbatim, and that only properly formatted DNS messages are passed.

There are two modes (so far) defined: DNS forwarder (dns clients point to a local gateway, which forwards to a remote gateway for sending to a DNS resolver); and transparent proxy (DNS packets are intercepted, passed to a local gateway, which sends them to the remote gateway, with original destination IP address etc. encoded, and used by the remote gateway as the destination).

DNS messages are NAT-friendly, so changes to IP or UDP headers do not impact them. Thus, SPARTACUS does not interfere with TSIG, SIG(0), or Eastlake Cookies.

This document describes the system, the components, and behavior, with examples.

Author's Note

Intended Status: Proposed Standard.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

DNS (The Domain Name System) has been deployed since the 1980's [RFC1033][RFC1034][RFC1035]. Since that time, some of the original Resource Record types have been made officially obsolete [RFC3425]. Some elements have been clarified [RFC2181][RFC2308]. New Resource Records have been added [RFC2136][RFC2845][RFC2930][RFC6891]. New definitions of bits in the header have arisen, in part due to DNSSEC's AD and CD bits [RFC4033][RFC4034][RFC4035][RFC5155].

This has resulted in now-outdated implementations of stateful devices (e.g. devices doing either NAT or packet inspection) interfering with end-to-end communication between DNS speakers. Old devices or implementations reject DNS packets that include these newer capabilities, features, or format changes.

At the same time, there has arisen a variety of other devices and systems whose deliberate function is to block, capture, or modify DNS traffic, for profit or for ideological reasons. Examples include hotel wifi systems, ISPs, and state actors.

Owing to the stateless nature of DNS over UDP, it is not possible to distinguish between deliberate and accidental sources of DNS interference.

1.1. Problem Statement

There is a need to provide ways of supporting incremental deployment of new DNS features, in such a way as to prevent deliberate and/or accidental interference in the communication between DNS speakers.

For example, DNS speakers could communicate over protected channels and with data integrity validation via DNSSEC. The foremost limitation is that the communication be over any other port/protocol combination than UDP port 53. Ideally, the choice should be an

encoding that is compatible with whatever port/protocol combination is selected (versus overloading the port/protocol with incompatible payloads).

There is a further need for the communications channel(s) to be standardized, and to not introduce further interoperability problems at the DNS protocol level. Independent implementations need to interoperate completely, to avoid merely pushing the compatibility problem around.

In order to solve these problems (individually and/or collectively), the SPARTACUS system has been developed.

1.2. Rationale

SPARTACUS (Secure, Private Aparatus for Resolution Transported Across Constraining and/or Unmaintained Systems), is a system for encoding and decoding DNS messages (the DNS payload of UDP or TCP packet streams).

The SPARTACUS system consists of bidirectional DNS gateways for transporting DNS over HTTP(S) using a JSON encoding scheme. This is intended to create "bridges" between DNS speakers; perhaps a better analogy would be "ferries", as there is no requirement for a tightly bound relationship between individual Client nodes and Server nodes.

Standardizing the JSON encoding used by SPARTACUS, is intended to ensure a greater likelihood of compatible, interoprable implementations.

The goal is to transport DNS messages from any Client implementation to any Server implementation.

Each gateway must be liberal in what it accepts (any valid DNS message conforming to the relevant RFCs, regardless of DNS implementation) and conservative in what it sends (all packets must parse correctly as DNS messages). In order to ensure forward compatibility, unknown Types and (in the case of OPT) sub-types, MUST be accepted and transported.

DNS messages MUST traverse the encode/decode process unaltered. The round-trip is designed to, and MUST be implemented to, preserve the entire DNS message's fidelity. This means a 1:1 binary match between input, encoding, decoding, and output. The lengths MUST match, and messages MUST be identical, bit for bit.

A secondary objective of the encoding in JSON is the use of the same names for data elements and structures as in the DNS RFCs. The idea

is to provide human-readable JSON encodings, for easier diagnostics during development, and when investigating operational issues.

1.3. Related Work

A variety of other work exists, and provided inspiration for the SPARTACUS work. This includes web/JSON DNS portals, for providing DNS query responses in JSON format, often with a "looking glass" functionality. FIXME format this list appropriately and decorate with words. END FIXME

- Multi-location DNS Looking Glass - Tool for performing DNS queries via RESTful interface in multiple locations, returning results in JSON format

- DNS Looking Glass - Tool for performing DNS queries via RESTful interface, returning results in JSON format

- DNS JSON - Source code project from circa 2009, partially developed but incomplete/abandoned

- DNSSEC-trigger[trigger] - embedded control function in NLnetlabs' Unbound resolver, for attempting DNS queries over TCP port 80 when DNSSEC problems are encountered

- Various other web-based DNS lookup tools

1.3.1. Comparison

There has been at least one previous effort to develop code for a DNS-JSON encoding, which appears to have been abandoned after one-way encoding was done, circa 2009. The project focused on presenting results to DNS queries in JSON format, with an intention to create a client gateway, which never materialized. The project can be found in two places ([JPF_jsondns] and [jsondns.org]). One major difference is that DNS query response status is converted to HTTP error codes, rather than being embedded in the JSON answer. This makes it unsuitable for bidirectional use. Only a few DNS type codes were implemented.

Another DNS JSON tool [fileformat.info], similarly focuses only on answers, with a limited number of type codes.

Yet another tool for looking up DNS via HTTP with JSON responses is the "dnsrest" [restdns.net]. It too focuses only on answer values, and is similarly not able to fully produce results that can be turned back into DNS answer packets.

The "DNS Looking Glass" [bortzmeyer.org], is primarily designed for returning DNS answer data. As such, it lacks encoding suitable for a bidirectional scheme. It is primarily focused on XML output, with JSON output organized around DNS resolution meta-data, plus answer data in a generic schema. (The schema itself is described in [draft-bortzmeyer-dns-json].)

The "Multilocation DNS Looking Glass" [dns-lg.com], uses a RESTful query mechanism of "node/qname/qtypename" to request the looking glass (LG) to perform a DNS lookup for the qname and qtype, and returns the response in a JSON format. The JSON format is generic, encapsulating all types as string data in presentation format, with a generic label of "rdata". This does not facilitate decoding easily, as the JSON scheme provides no information for parsing the rdata field. The type (qtype for the query, or type for answer/authority/additional) is in string (symbolic) form, and the elements are objects and thus in unordered lists. The JSON scheme is fine for one-way encoding for human readability, but not suitable for two-way conversion back into DNS.

DNSSEC-trigger[trigger] can only be used in environments that use NLnetlabs' Unbound resolver, or where Unbound can be deployed as a replacement for existing recursive resolvers and/or stub resolvers.

A variety of other web lookup tools exist, predominantly producing DNS validation (zone structure and hierarchy), maps, meta-data, or literal output from the 'dig' tool, in formats as varied as the purposes of the tools. Dig output, while being reasonably deterministic, is not sufficiently well-formed as to facilitate "screen scraping" as a parsing method.

2. Requirements

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. System Overview

The SPARTACUS system is designed to improve the reliability and security of the DNS system, by providing the means to transport DNS traffic across segments of the Internet. The goal is to bypass problem areas which interfere with DNS communications, regardless of root cause of the interference.

Some familiarity with the DNS protocol is assumed.

3.1. System Elements

The particular system elements used will differ, based on the mode of operation of the Client. Clients may request the use of particular resolvers via additional intra-element signalling.

3.1.1. Node Types

Base node types are the following:

- Standalone SPARTACUS Client forwarder
- Transparent SPARTACUS proxy Client
- Standalone SPARTACUS Server
- Apache module-based SPARTACUS Server
- Stub resolver
- External recursive resolver
- Client-side recursive resolver
- External authority server

Future node types are expected to include:

- Browser-integrated SPARTACUS client and stub resolver
- Mobile-device SPARTACUS client and stub resolver (with exposed getdns API)
- SMTP-integrated SPARTACUS client and stub resolver

3.2. System Modes

The system has two modes of operation:

- DNS Forwarder - an opaque mode of operation, the Client/Server pair act collectively as a single DNS forwarder.
- Transparent Proxy - In this mode, regular DNS traffic is diverted by unspecified means to the SPARTACUS Client.

Additional intra-element signalling facilitates Clients requesting particular resolvers' (recursive or authoritative) use.

3.2.1. Details on DNS Forwarder mode

The Server is configured to use a particular DNS recursive resolver, with the optional ability to support Client-requested resolver(s) via in-band signaling. If present, the Client-requested resolver IP address is passed as an EDNS OPT value. The Server, if it is configured to honor requested resolvers, uses this IP address instead of the default.

Example: Problem caused by firewalls that do not support DNSSEC:

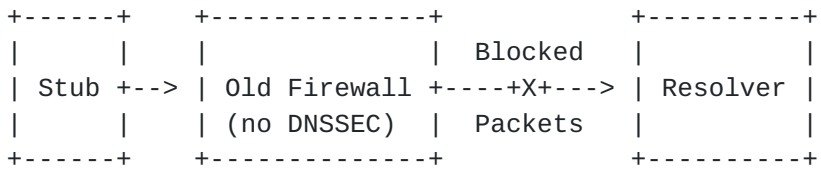

Figure 1

Example: How the stub client sees the SPARTACUS Client/Server pair, in the opaque forwarder configuration:

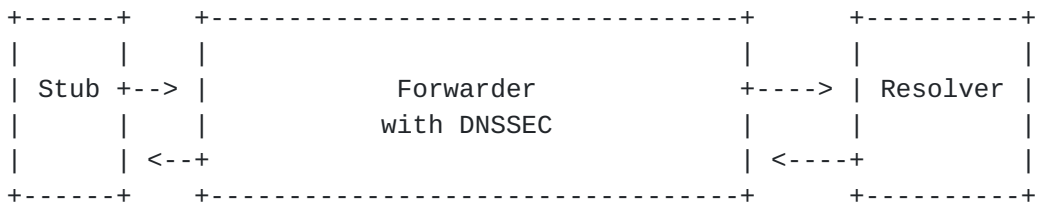

Figure 2

Example: How the Client/Server pair actually operates:

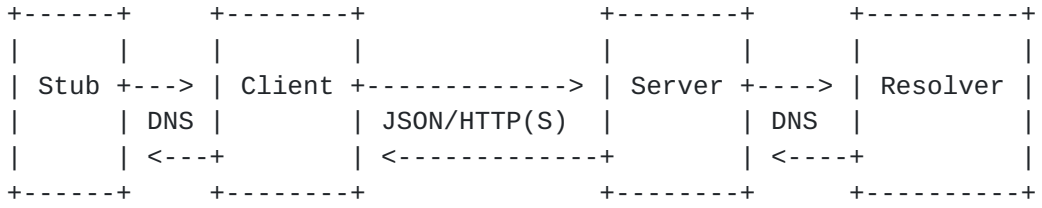

Figure 3

Example: How the Client/Server bypass the old firewall:

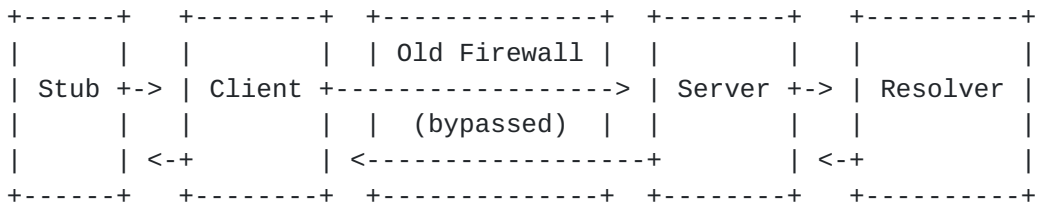

Figure 4

3.2.2. Details on Transparent Proxy mode

Transparent Proxy mode supports transport of stub to recursive traffic (all with the same destination IP address).

Transparent Proxy mode also supports use by a recursive resolver, to handle recursive-to-authoritative traffic (with different destination IP addresses per query).

From the perspective of the DNS client (stub or recursive), it appears that the DNS query packet went to some IP address, and the reply came back directly.

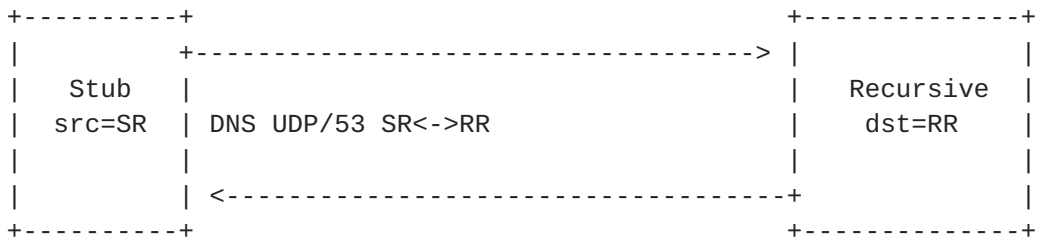

Figure 5

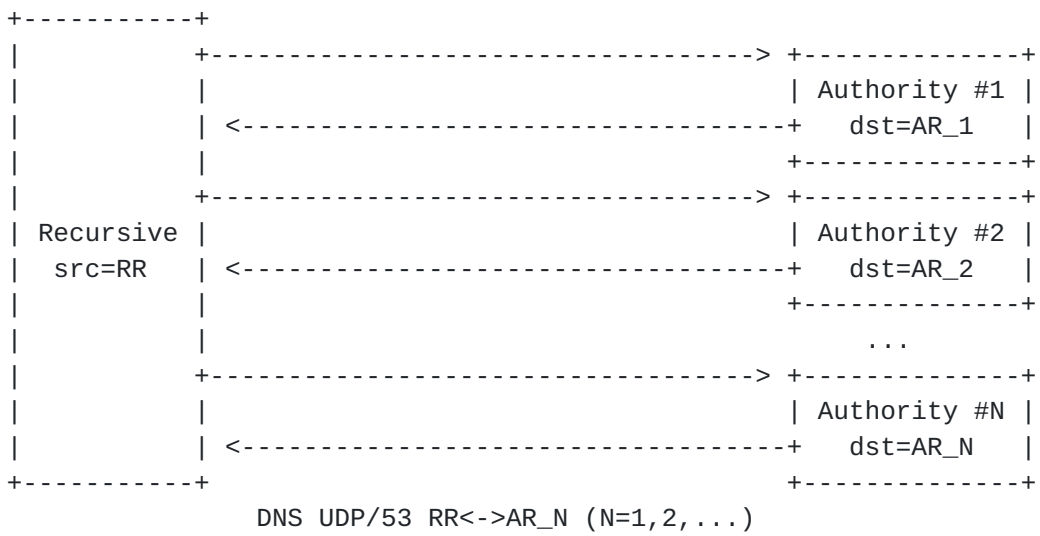

Figure 6

In both use cases, the original IP destination is encoded as an EDNS OPT value, and the DNS message (encoded as JSON) is sent to the SPARTACUS Server. The Server sends the DNS message to the original IP destination, with the SPARTACUS Server's IP address as the source. The resulting answer DNS message is sent to the Client, which changes the reply source IP address to the original destination IP address.

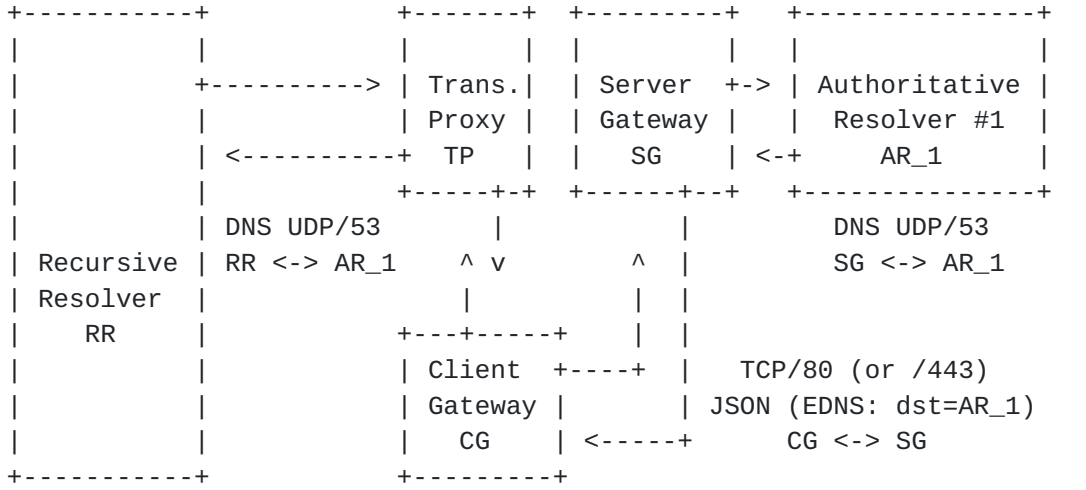

Figure 7

The only practical difference is that some intermediate devices see JSON/HTTP(S) instead of DNS/UDP traffic. For some of those devices, this is in fact the purpose of SPARTACUS - preventing those devices from inspecting the DNS traffic in a problematic manner.

3.3. Interoperability

The purpose of this document is to ensure that independent implementations of Client(s) and Server(s) can interoperate, so long as each is permitted to interoprate with the other.

It is not required that Servers be operated in a completely "open" manner. However, the more open Servers there are, the greater the benefit. Like any web-based service, care should be given towards managing available resources on a Server. In all likelihood, this resource management may be most effectively handled via the web server's own service management system.

3.3.1. In-scope and out-of-scope

The following items are out-of-scope, from an interoperability standpoint.

This means that individual implementations may make independent design decisions, without impacting interoperability.

- Choice(s) of default resolver (on Server)
- Server-side DNS retry and time-out values
- How Client(s) select Servers

The following items are in-scope, from an interoperability standpoint.

- JSON encoding
- How to signal non-support of requested resolver(s)
- How to signal "no response" (timeout) on Server-resolver traffic
- Signalling/encoding of default,requested, and actually-used resolvers
- Stripping of EDNS OPT private values
- Stripping of synthesized EDNS OPT record

The following items are optional, from an interoperability standpoint.

- Whether and how to do edns-client-subnet (on Client)

- Whether to use TLS (HTTPS) on Client-Server traffic
- Whether to honor requested resolvers (on Server)
- Whether to support Transparent Proxy mode (on the Client)
- Whether to do DNSSEC validation
- Whether to do PKI validation of SSL certificates (if HTTPS is used and CA-issued certs used)
- Whether to do DANE validation of SSL certificates (if HTTPS is used and TLSA records exist)
- Whether IPv4 or IPv6 is supported

4. Interactions and Behavior

The Client Gateway needs to make informed decisions about Server Gateways to use. Client Gateways may use pre-configured (static) gateways, or may employ any number of strategies for selection of Server Gateways.

In order to enble Client-controlled Server Selection, each Server Gateway needs to advise the Client about default and actual DNS Servers used. The Client optionally requests DNS Server(s) that the Server should use. If present, the Server includes that in the response.

The SPARTACUS client/server interaction occurs over TCP rather than UDP. As such, other than TCP-based failures (RST aka "reset" for example), every query MUST get a response (owing to the HTTP POST standards).

Since the Server Gateway is performing DNS resolution using UDP transport, it is possible that network packet loss may occur, resulting in unanswered queries.

Also, there are reasons other than network-based packet loss that can result in unanswered queries. DNS resolvers must attempt to infer what causes queries to not be answered.

It is also possible that various other failure modes could occur, which need to be handled on the basis of the nature of the failure.

Each of these is addressed in separate sections below.

4.1. DNS Gateway Encodings

The three DNS Server values (default, requested, actual) are communicated via EDNS OPT type-length-value (TLV) tuples, using three distinct types. Pre-standard experimental values are presently being used. IANA will need to assign permanent OPT Type values for these three type codes.

In order to ensure that the EDNS OPT record is only returned to the original DNS client if it existing in the query, it is necessary to identify cases where the DNS Server value encoding resulted in a "new" OPT record, rather than being added to an existing record. In such cases, an additional OPT TLV type is required, and is added to the OPT record. A fourth OPT Type value needs to be assigned by IANA for this purpose.

The new OPT codes are used to enable the Client and Server to maintain all communication details inside the DNS message itself. This simplifies the design, implementation, and operation of Clients and Servers, and ensures forward/backward compatibility. OPT codes specific to the Client-Server communication MUST be removed prior to forwarding of DNS messages to DNS Clients and Servers. If the EDNS OPT RR is synthesized (added to the DNS message), it MUST be removed.

4.2. UDP Packet Loss

In cases where the Server Gateway did not get a response from the DNS Server, it needs to signal this back to the client. It needs to do this so that the proper Client state is established. This prevents time-out based (undefined) behavior on the Client from being triggered. The Server needs to "pass along" packet loss status to the Client to trigger well-defined Client behavior.

The mechanism is to use a Private EDNS OPT type/length/value (TLV), with the original Question echoed back (to associate with the Query). When receiving this TLV, the Client will treat this as a lost UDP packet, and MUST NOT send back any UDP packet. The UDP client is responsible for handling this lost UDP packet, per the DNS protocol.

4.3. Malformed UDP response

The malformed UDP packet may not be legitimate. To be conservative, this condition is signaled back to the client, and the (actual) received UDP packet is rejected/dropped. This is treated by the DNS client as a lost UDP packet.

4.4. DNSSEC Validation Failure

If DNSSEC validation fails, the presumption needs to be made that the failure is deliberate. The DNSSEC standards call for "SRVFAIL" responses, so that is what a compliant implementation MUST return to the UDP client.

If the Client and/or Server does DNSSEC Validation, it MUST correctly implement Validation signalling via the AD and CD bits.

In other words, it MUST return the answer regardless of Validation if the CD bit is set, and it MUST set the AD bit if Validation succeeds, regardless of the presence and/or state of EDNS bit DO.

5. Client-Server Selection and Topology Examples

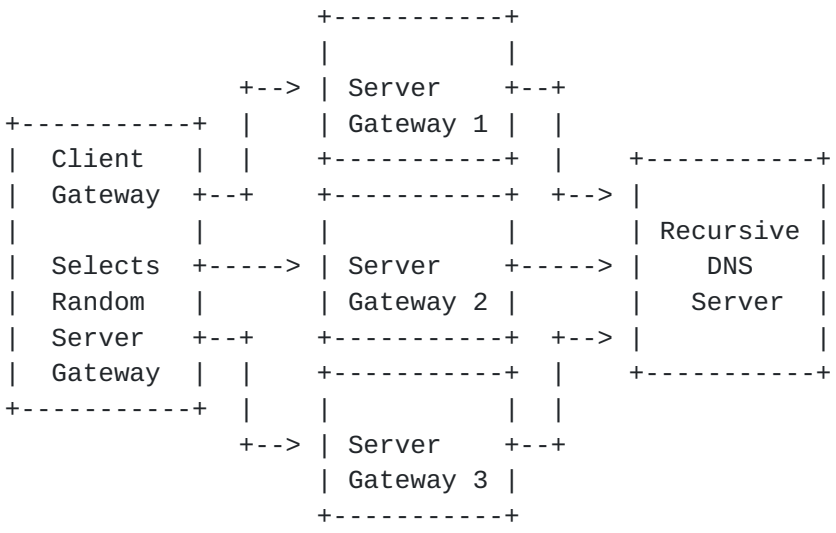

Figure 8

Figure 8 shows the same Recursive DNS Server being used, via multiple Server Gateways. There are several benefits to doing this; they include distributing load among multiple Server Gateways, and reducing the amount of DNS traffic going via any single Server Gateway. This limits the impact of the compromise of any single Server Gateway, or of any single Server Certificate compromise.

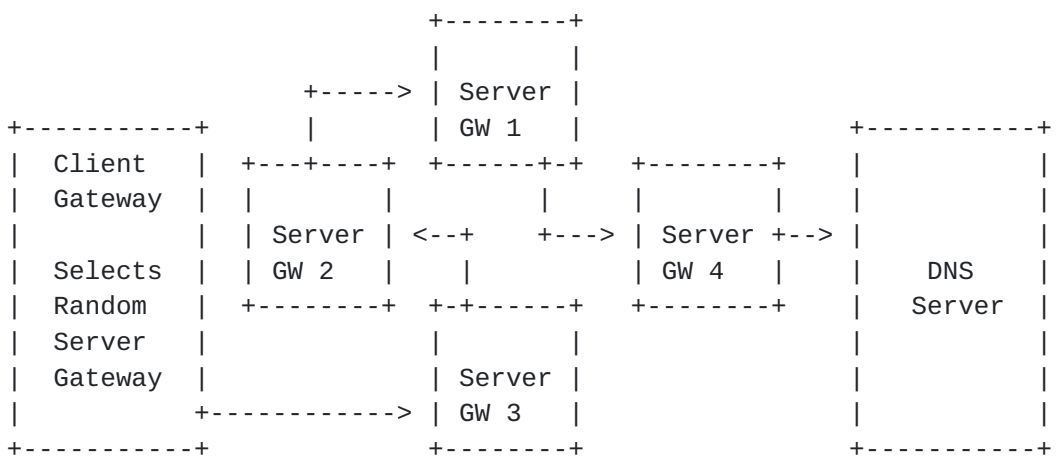

Figure 9

Figure 9 illustrates a path where more than one Server Gateway is traversed during resolution. The objective here is to disassociate the IDENTITY of the client from the CONTENT of the query/answer. The association is only made directly on the first Server Gateway (and only with respect to the Client Gateway). The actual association of the source UDP client is only done on the Client Gateway itself, which may or may not provide further privacy. Since there is more than one Server-Server hop, this significantly reduces the ability to infer associations between Query/Response and Client Gateways.

It should be noted that this looks very much like TOR (The Onion Router), applied to JSON-encoded UDP DNS traffic. There is a proposal for DNS privacy enhancements that applies a similar techique, directly on UDP-based DNS queries/answers. FIXME add xref here to reference to the appropriate Internet Draft. END FIXME

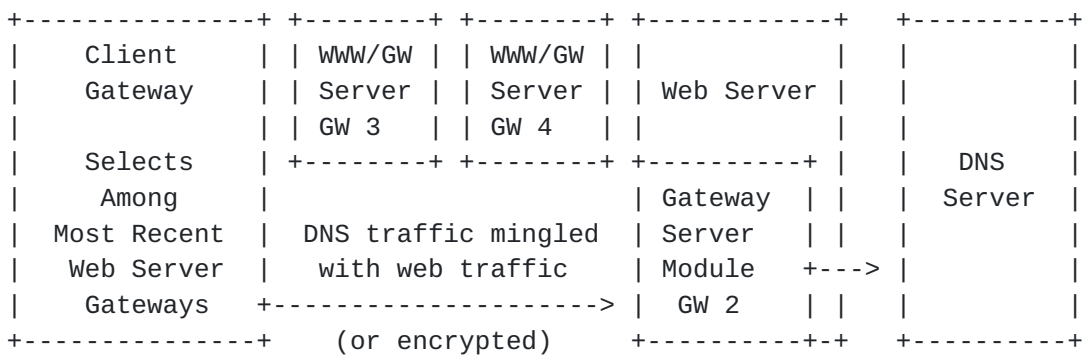

Figure 10

Figure 10 illustrates one query/response when the client is attempting to use something similar to steganography to preserve privacy. In this context, the privacy against passive monitoring is

achieved by using un-blocked web servers which are also Server Gateways. A MitM adversary cannot easily block this traffic without blocking the entire site, or by inspecting every flow to/from the site. A passive observer would similarly need to inspect all flows to find the embedded, encoded DNS traffic. The DNS traffic would be nearly indistinguishable from regular HTTP traffic.

Note that the use of TLS to protect the Client-Server traffic would make it impossible to distinguish the DNS traffic from the other web trafficin this situation. Combining this "tag-along" with TLS provides both strong privacy and strong security.

5.1. Mixed Traffic Walk-Through

Suppose a client were to visit web sites "a" through "j" sequentially, i.e. a,b,c,d,e,f,g,h,i,j. Suppose some of those were also Server Gateways, represented by upper case (vs lower case for web sites without Server Gateway capabilities). Thus the sequence would be A,b,C,D,e,f,g,H,I,j. If the Client Gateway chose a Server Gateway randomly from among the last four web sites visited, the sequence of events after visiting A through D, would look like:

- Select Server from set {A,C,D}, look up "e". Visit "e".
- Select Server from set {C,D}, look up "f". Visit "f".
- Select Server from set {C,D}, look up "g". Visit "g".
- Select Server from set {D}, look up "h". Visit "h".
- Select Server from set {D,H}, look up "i". Visit "i".
- Select Server from set {H,I}, look up "j". Visit "j".

An observer close to the Client would see traffic within a given time window, only to the same set of Web servers. An observer close to any of the Web servers would only see traffic from a given client, for a small interval of time after the first visit.

6. Security Considerations

(None per se.) Need to list considerations etc.

7. IANA Considerations

This document will eventually contain IANA-specific material.

8. Acknowledgements

To be added later.

9. References

9.1. Normative References

[RFC1033] Lottor, M., "Domain administrators operations guide", RFC 1033, November 1987.

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, November 1987.

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, November 1987.

[RFC2136] Vixie, P., Thomson, S., Rekhter, Y., and J. Bound, "Dynamic Updates in the Domain Name System (DNS UPDATE)", RFC 2136, April 1997.

[RFC2181] Elz, R. and R. Bush, "Clarifications to the DNS Specification", RFC 2181, July 1997.

[RFC2308] Andrews, M., "Negative Caching of DNS Queries (DNS NCACHE)", RFC 2308, March 1998.

[RFC2845] Vixie, P., Gudmundsson, O., Eastlake, D., and B. Wellington, "Secret Key Transaction Authentication for DNS (TSIG)", RFC 2845, May 2000.

[RFC2930] Eastlake, D., "Secret Key Establishment for DNS (TKEY RR)", RFC 2930, September 2000.

[RFC3425] Lawrence, D., "Obsoleting IQUERY", RFC 3425, November 2002.

[RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, March 2005.

[RFC4034] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Resource Records for the DNS Security Extensions", RFC 4034, March 2005.

[RFC4035] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Protocol Modifications for the DNS Security Extensions", RFC 4035, March 2005.

[RFC5155] Laurie, B., Sisson, G., Arends, R., and D. Blacka, "DNS Security (DNSSEC) Hashed Authenticated Denial of Existence", RFC 5155, March 2008.

[RFC6891] Damas, J., Graff, M., and P. Vixie, "Extension Mechanisms for DNS (EDNS(0))", STD 75, RFC 6891, April 2013.

9.2. Informative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[JPF_jsondns]
"DNS over HTTP", <http://github.com/jpf/jsondns>.

[jsondns.org]
Franusic, J., "Query DNS via REST", <http://jsondns.org/>.

[fileformat.info]
Marcuse, A., "DNS in client-side JavaScript", <http://www.fileformat.info/tool/rest/dns-json.htm>.

[restdns.net]
"REST-DNS", <http://restdns.net/>.

[bortzmeyer.org]
Bortzmeyer, S., "DNS Looking Glass", <http://www.bortzmeyer.org/dns-lg.html>.

[draft-bortzmeyer-dns-json]
Bortzmeyer, S., "DNS in JSON", <http://tools.ietf.org/html/draft-bortzmeyer-dns-json-01>.

[dns-lg.com]
Cambus, F., "Multilocation DNS Looking Glass", <http://www.dns-lg.com/>.

[trigger] NLnet Labs, "Dnssec-Trigger", <http://www.nlnetlabs.nl/projects/dnssec-trigger/>.

Appendix A. DNS Message Encoding Examples

The entire encoding of pairs of DNS messages follows. For each pair,the first is the query, and the second is the response.

A.1. Simple Query/Answer, No EDNS or DNS Server

Query encoded in JSON:

```
"PACKET (RFC 1035)" : [
"ROLE" : "client",
"DSIZE" : 26,
"DICTIONARY" : [
"example",
"com",
""
],
"DSIZE2" : 26,
"Header" : [
"ID" : 42,
"HFlags" : [
"QR" : false,
"Opcode" : [ "Query" : 0 ],
"AA" : false,
"TC" : false,
"RD" : true,
"RA" : false,
"Z" : false,
"AD" : false,
"CD" : false,
"RCODE" : [ "NoError (RFC 1035)" : 0 ]

],
"QDCOUNT" : 1,
"ANCOUNT" : 0,
"NSCOUNT" : 0,
"ARCOUNT" : 0
],
"Question" : [
"QUESTION (RFC 1035)" : [
"QNAME" : [ "example.com." : 0 ],
"QTYPE" : [ "A" : 1 ],
"QCLASS" : [ "IN" : 1 ]
]
]
]
```

Response encoded in JSON:

```
"PACKET (RFC 1035)" : [
"ROLE" : "client",
"DSIZE" : 33,
"DICTIONARY" : [
```

```
"example",
"com",
"",
"@0"
],
"DSIZE2" : 33,
"Header" : [
"ID" : 42,
"HFlags" : [
"QR" : true,
"Opcode" : [ "Query" : 0 ],
"AA" : false,
"TC" : false,
"RD" : true,
"RA" : true,
"Z" : false,
"AD" : false,
"CD" : false,
"RCODE" : [ "NoError (RFC 1035)" : 0 ]

],
"QDCOUNT" : 1,
"ANCOUNT" : 1,
"NSCOUNT" : 0,
"ARCOUNT" : 0
],
"Question" : [
"QUESTION (RFC 1035)" : [
"QNAME" : [ "example.com." : 0 ],
"QTYPE" : [ "A" : 1 ],
"QCLASS" : [ "IN" : 1 ]
]
],
"Answer" : [
"RR" : [
"NAME" : [ "example.com." : 0 ],
"TYPE" : [ "A" : 1 ],
"CLASS" : [ "IN" : 1 ],
"TTL" : 5218,
"RDLENGTH" : 4,
"RDATA" : [
"A" : [
"Address" : "93.184.216.119"
]
]
]
]
]
```

A.2. Simple Query/Answer, EDNS, no DNS Server

Query encoded in JSON:

```
"PACKET (RFC 1035)" : [
"ROLE" : "client",
"DSIZE" : 31,
"DICTIONARY" : [
"example",
"com",
"",
""
],
"DSIZE2" : 31,
"Header" : [
"ID" : 42,
"HFlags" : [
"QR" : false,
"Opcode" : [ "Query" : 0 ],
"AA" : false,
"TC" : false,
"RD" : true,
"RA" : false,
"Z" : false,
"AD" : false,
"CD" : false,
"RCODE" : [ "NoError (RFC 1035)" : 0 ]

],
"QDCOUNT" : 1,
"ANCOUNT" : 0,
"NSCOUNT" : 0,
"ARCOUNT" : 1
],
"Question" : [
"QUESTION (RFC 1035)" : [
"QNAME" : [ "example.com." : 0 ],
"QTYPE" : [ "A" : 1 ],
"QCLASS" : [ "IN" : 1 ]
]
],
"Additional" : [
"RR" : [
"NAME" : [ "." : 3 ],
"TYPE" : [ "OPT" : 41 ],
"Field3" : [
"UDPSIZEFIELD" : [
"UDPSIZE" : 1500
```

```
]
],
"Field4" : [
"Extended_RCode_Flags" : [
"ERCFlagbits" : [
"RCode" : 0,
"Version" : 0,
"DO" : false,
"Resv" : 0

]
]
],
"RDLENGTH" : 0,
"RDATA" : [
"OPT (RFC 6891)" : [
"TLV_LIST" : [

]
]
]
]
]
]
```

Response encoded in JSON:

```
"PACKET (RFC 1035)" : [
"ROLE" : "client",
"DSIZE" : 38,
"DICTIONARY" : [
"example",
"com",
"",
"@0",
""
],
"DSIZE2" : 38,
"Header" : [
"ID" : 42,
"HFlags" : [
"QR" : true,
"Opcode" : [ "Query" : 0 ],
"AA" : false,
"TC" : false,
"RD" : true,
"RA" : true,
"Z" : false,
```

```
"AD" : false,
"CD" : false,
"RCODE" : [ "NoError (RFC 1035)" : 0 ]

],
"QDCOUNT" : 1,
"ANCOUNT" : 1,
"NSCOUNT" : 0,
"ARCOUNT" : 1
],
"Question" : [
"QUESTION (RFC 1035)" : [
"QNAME" : [ "example.com." : 0 ],
"QTYPE" : [ "A" : 1 ],
"QCLASS" : [ "IN" : 1 ]
]
],
"Answer" : [
"RR" : [
"NAME" : [ "example.com." : 0 ],
"TYPE" : [ "A" : 1 ],
"CLASS" : [ "IN" : 1 ],
"TTL" : 4865,
"RDLENGTH" : 4,
"RDATA" : [
"A" : [
"Address" : "93.184.216.119"
]
]
]
],
"Additional" : [
"RR" : [
"NAME" : [ "." : 4 ],
"TYPE" : [ "OPT" : 41 ],
"Field3" : [
"UDPSIZEFIELD" : [
"UDPSIZE" : 4000
]
],
"Field4" : [
"Extended_RCode_Flags" : [
"ERCFlagbits" : [
"RCode" : 0,
"Version" : 0,
"DO" : false,
"Resv" : 0
```

```
]
]
],
"RDLENGTH" : 0,
"RDATA" : [
"OPT (RFC 6891)" : [
"TLV_LIST" : [

]
]
]
]
]
]
```

A.3. Simple Query/Answer, no EDNS, with DNS Server

Query encoded in JSON:

```
"PACKET (RFC 1035)" : [
"ROLE" : "client",
"DSIZE" : 31,
"DICTIONARY" : [
"example",
"com",
"",
""
],
"DSIZE2" : 31,
"Header" : [
"ID" : 42,
"HFlags" : [
"QR" : false,
"Opcode" : [ "Query" : 0 ],
"AA" : false,
"TC" : false,
"RD" : true,
"RA" : false,
"Z" : false,
"AD" : false,
"CD" : false,
"RCODE" : [ "NoError (RFC 1035)" : 0 ]

],
"QDCOUNT" : 1,
"ANCOUNT" : 0,
"NSCOUNT" : 0,
"ARCOUNT" : 1
```

```
   ],
   "Question" : [
   "QUESTION (RFC 1035)" : [
   "QNAME" : [ "example.com." : 0 ],
   "QTYPE" : [ "A" : 1 ],
   "QCLASS" : [ "IN" : 1 ]
   ]
   ],
   "Additional" : [
   "RR" : [
   "NAME" : [ "." : 3 ],
   "TYPE" : [ "OPT" : 41 ],
   "Field3" : [
   "UDPSIZEFIELD" : [
   "UDPSIZE" : 1500
   ]
   ],
   "Field4" : [
   "Extended_RCode_Flags" : [
   "ERCFlagbits" : [
   "RCode" : 0,
   "Version" : 0,
   "DO" : false,
   "Resv" : 0

   ]
   ]
   ],
   "RDLENGTH" : 19,
   "RDATA" : [
   "OPT (RFC 6891)" : [
   "TLV_LIST" : [
   "TLV" : [
   "TYPE" : [ "PrivateType65500" : 65500 ],
   "Len" : 13,
   "Data" : [
   "PrivateType65500" : [
   "GW_NAME" : [ "10:10" , "198.41.1.1" ]

   ]
   ]
   ],
   "TLV" : [
   "TYPE" : [ "PrivateType65510" : 65510 ],
   "Len" : 0,
   "Data" : [
   ]
   ],
```

```
]
]
]
]
]
]
```

Response encoded in JSON:

```
"PACKET (RFC 1035)" : [
"ROLE" : "client",
"DSIZE" : 38,
"DICTIONARY" : [
"example",
"com",
"",
"@0",
""
],
"DSIZE2" : 38,
"Header" : [
"ID" : 42,
"HFlags" : [
"QR" : true,
"Opcode" : [ "Query" : 0 ],
"AA" : false,
"TC" : false,
"RD" : true,
"RA" : true,
"Z" : false,
"AD" : true,
"CD" : false,
"RCODE" : [ "NoError (RFC 1035)" : 0 ]

],
"QDCOUNT" : 1,
"ANCOUNT" : 1,
"NSCOUNT" : 0,
"ARCOUNT" : 1
],
"Question" : [
"QUESTION (RFC 1035)" : [
"QNAME" : [ "example.com." : 0 ],
"QTYPE" : [ "A" : 1 ],
"QCLASS" : [ "IN" : 1 ]
]
],
"Answer" : [
```

```
"RR" : [
"NAME" : [ "example.com." : 0 ],
"TYPE" : [ "A" : 1 ],
"CLASS" : [ "IN" : 1 ],
"TTL" : 4084,
"RDLENGTH" : 4,
"RDATA" : [
"A" : [
"Address" : "93.184.216.119"
]
]
]
],
"Additional" : [
"RR" : [
"NAME" : [ "." : 4 ],
"TYPE" : [ "OPT" : 41 ],
"Field3" : [
"UDPSIZEFIELD" : [
"UDPSIZE" : 512
]
],
"Field4" : [
"Extended_RCode_Flags" : [
"ERCFlagbits" : [
"RCode" : 0,
"Version" : 0,
"DO" : false,
"Resv" : 0

]
]
],
"RDLENGTH" : 19,
"RDATA" : [
"OPT (RFC 6891)" : [
"TLV_LIST" : [
"TLV" : [
"TYPE" : [ "PrivateType65500" : 65500 ],
"Len" : 13,
"Data" : [
"PrivateType65500" : [
"GW_NAME" : [ "10:10" , "198.41.1.1" ]

]
]
],
"TLV" : [
```

```
"TYPE" : [ "PrivateType65510" : 65510 ],
"Len" : 0,
"Data" : [
]
],

]
]
]
]
]
]
```

A.4. Simple Query/Answer, with EDNS and DNS Server

Query encoded in JSON:

```
"PACKET (RFC 1035)" : [
"ROLE" : "client",
"DSIZE" : 31,
"DICTIONARY" : [
"example",
"com",
"",
""
],
"DSIZE2" : 31,
"Header" : [
"ID" : 42,
"HFlags" : [
"QR" : false,
"Opcode" : [ "Query" : 0 ],
"AA" : false,
"TC" : false,
"RD" : true,
"RA" : false,
"Z" : false,
"AD" : false,
"CD" : false,
"RCODE" : [ "NoError (RFC 1035)" : 0 ]

],
"QDCOUNT" : 1,
"ANCOUNT" : 0,
"NSCOUNT" : 0,
"ARCOUNT" : 1
],
"Question" : [
```

```
"QUESTION (RFC 1035)" : [
"QNAME" : [ "example.com." : 0 ],
"QTYPE" : [ "A" : 1 ],
"QCLASS" : [ "IN" : 1 ]
]
],
"Additional" : [
"RR" : [
"NAME" : [ "." : 3 ],
"TYPE" : [ "OPT" : 41 ],
"Field3" : [
"UDPSIZEFIELD" : [
"UDPSIZE" : 1500
]
],
"Field4" : [
"Extended_RCode_Flags" : [
"ERCFlagbits" : [
"RCode" : 0,
"Version" : 0,
"DO" : false,
"Resv" : 0

]
]
],
"RDLENGTH" : 15,
"RDATA" : [
"OPT (RFC 6891)" : [
"TLV_LIST" : [
"TLV" : [
"TYPE" : [ "PrivateType65500" : 65500 ],
"Len" : 13,
"Data" : [
"PrivateType65500" : [
"GW_NAME" : [ "10:10" , "198.41.1.1" ]

]
]
],

]
]
]
]
]
]
```

Response encoded in JSON:

```
"PACKET (RFC 1035)" : [
"ROLE" : "client",
"DSIZE" : 38,
"DICTIONARY" : [
"example",
"com",
"",
"@0",
""
],
"DSIZE2" : 38,
"Header" : [
"ID" : 42,
"HFlags" : [
"QR" : true,
"Opcode" : [ "Query" : 0 ],
"AA" : false,
"TC" : false,
"RD" : true,
"RA" : true,
"Z" : false,
"AD" : true,
"CD" : false,
"RCODE" : [ "NoError (RFC 1035)" : 0 ]

],
"QDCOUNT" : 1,
"ANCOUNT" : 1,
"NSCOUNT" : 0,
"ARCOUNT" : 1
],
"Question" : [
"QUESTION (RFC 1035)" : [
"QNAME" : [ "example.com." : 0 ],
"QTYPE" : [ "A" : 1 ],
"QCLASS" : [ "IN" : 1 ]
]
],
"Answer" : [
"RR" : [
"NAME" : [ "example.com." : 0 ],
"TYPE" : [ "A" : 1 ],
"CLASS" : [ "IN" : 1 ],
"TTL" : 4084,
"RDLENGTH" : 4,
"RDATA" : [
```

```
   "A" : [
   "Address" : "93.184.216.119"
   ]
   ]
   ]
   ],
   "Additional" : [
   "RR" : [
   "NAME" : [ "." : 4 ],
   "TYPE" : [ "OPT" : 41 ],
   "Field3" : [
   "UDPSIZEFIELD" : [
   "UDPSIZE" : 512
   ]
   ],
   "Field4" : [
   "Extended_RCode_Flags" : [
   "ERCFlagbits" : [
   "RCode" : 0,
   "Version" : 0,
   "DO" : false,
   "Resv" : 0

   ]
   ]
   ],
   "RDLENGTH" : 15,
   "RDATA" : [
   "OPT (RFC 6891)" : [
   "TLV_LIST" : [
   "TLV" : [
   "TYPE" : [ "PrivateType65500" : 65500 ],
   "Len" : 13,
   "Data" : [
   "PrivateType65500" : [
   "GW_NAME" : [ "10:10" , "198.41.1.1" ]

   ]
   ]
   ],

   ]
   ]
   ]
   ]
   ]
   ]
```

Appendix B. Server Gateway HTML code

The entire HTML document needed on the Server for the Client to send/ receive JSON-encoded DNS messages follows:

```
<html>
<body>
<form action="cgi-bin/json-resolver2.pl" method="POST">
<P>
<TEXTAREA name="query" rows="20" cols="80">
</TEXTAREA>
<INPUT type="submit" value="Send"><INPUT type="reset">
</P>
</form>
</body>
</html>
```

The "action" target needs to exist and be executable, and ideally be performance-optimized (e.g. via use of mod_perl).

Appendix C. Server Gateway HTTP POST Handler Pseudo-code

The following pseudo-code illustrates the high-level behavior of the HTML handler for the Server.

The handler is passed the contents of the TEXTAREA, which will be the JSON-encoded DNS message.

```
// initialize parser etc.
// set up socket for UDP query/response to default Resolver
// set up socket for UDP query/response to client-supplied Resolver
// extract JSON-encoded DNS message from HTTP POST variable 'query'
// save original Query-ID, assign new Query-ID (to avoid collisions)
// decode DNS message (into DNS wire format)
// if DNS message has OPT Resource Record
//   if OPT has Client-supplied Resolver option
//     extract Resolver value
//     delete Resolver option from OPT
//   endif
//   if OPT was synthesized by Client
//     delete OPT Resource Record
//   endif
//   send DNS message to Client-specified Resolver
// else
//   send DNS message to default Resolver
// endif
// wait for response or timeout
// if timeout && retry-count < max-retry-count
//   resend DNS message
// elsif timeout && retry-count >= max-retry-count
//   send "retry-count-exceeded" via OPT (synthesized if necessary)
// else
//   set DNS answer's Query-ID value to original Query-ID
//   encode DNS answer
//   send JSON-encoded answer to Client
// endif
```

Appendix D. Client Gateway Pseudo-code

The following pseudo-code illustrates the high-level behavior of the Client.

The Client in this example is pre-configured with a single Server Gateway's address.

```
   // initialize parser etc.
   // set up socket for UDP query/response (Listener)
   // set up HTTP connection to Server Gateway
   // do an HTTP "GET" to the predefined URL of the Server HTML code
   // extract HTML elements needed: handler, variable name
   // loop forever:
   //   listen for DNS query packet
   //   fork (to handle this packet)
   //   if child
   //     save old DNS Query-ID, set new Query-ID
   //     if Use-Supplied-Resolver
   //       if exits OPT
   //         add client-supplied-resolver to OPT
   //       else
   //         synthesize OPT and add client-supplied-resolver
   //       endif
   //     endif
   //     encode DNS message (into JSON)
   //     write HTTP POST onto socket
   //     wait for HTTP response
   //     extract JSON-encoded answer from HTTP
   //     decode DNS answer (from JSON)
   //     if OPT
   //       if OPT.option is error condition
   //         drop answer and continue loop forever:
   //       elsif OPT synthesized
   //         delete OPT
   //       elsif OPT.option SPARTACUS-specific value
   //         delete option
   //       endif
   //     endif
   //     set answer.Query-ID to saved value
   //     send answer to sender
   //   end-of-loop
```

Author's Address

Brian Dickson
12047B 36th Ave NE
Seattle, wA 98125

Email: brian.peter.dickson@gmail.com