Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: June 11, 2011

P. Hoffman
VPN Consortium
December 8, 2010

Wrapping Last-Hop DNS for Traffic Protection
draft-hoffman-dns-last-hop-01

Abstract

DNS queries from one resolver to an upstream resolver are often run over connections with no protection of any kind. The stub resolvers that initiate queries for an application are ofte called "last-hop", and their queries go to trusted recursive resolvers. The connection between last-hop resolvers and their upstream resolver is currently susceptible to both malicious and unintentional alteration that prevents the querying resolver from being sure that the results it receives are valid. Some middleboxes can prevent a stub resolver, even one that does DNSSEC validation, from getting enough information to validate a response. Further, a non-validating stub resolver is susceptible to active attacks in which the results are purposely altered.

The protocol described in this document provides a method to avoid these problems and thus make resolution significantly more secure. This protocol can be used between any two DNS resolvers, but the focus of this document is on last-hop resolvers.

1. Introduction

In the original specification of DNS (which is broadly defined in [RFC1034], [RFC1035], and others), queries travel between a resolver and a server without any cryptographic integrity protection. DNSSEC (which is broadly defined in RFCs 4033, 4034, and 4035 ([RFC4033], [RFC4034], and [RFC4035]) adds integrity protection of the response message and thus allows a validating resolver to verify that a message has not been altered in transit.

RFCs 1034, 1035, and 4033 have many definitions that apply to the discussion here. The term "resolver" is defined in RFC 1034. A "stub resolver" is a resolver that sends queries in recursive mode to a recursive resolver, but does not do recursion itself. In the context of these definitions, "security-aware" means supports the EDNS0 message size extension from [RFC2671] and the DO bit from [RFC3225], and supports the RR types and message header bits defined in the DNSSEC documents. A "security-aware resolver" is an entity that acts as a resolver and that understands DNSSEC. Two types of stub resolvers are discussed in this document. A "validating stub resolver" is a security-aware resolver that sends queries in recursive mode but that performs signature validation on its own rather than just blindly trusting an upstream security-aware recursive name server. A "non-validating stub resolver" is a security-aware resolver that trusts one or more security-aware recursive name servers to perform DNSSEC validation on its behalf.

In typical deployments of DNS, there is a stub resolver that is the original source of the DNS query. This query usually traverses one or more recursive DNS servers on its way to the authoritative server for the data. DNSSEC provides a mode (using the AD bit) in which a security-aware but non-validating resolver (usually a stub resolver) may trust that the data it receives has been validated, but only if the connection between it and the upstream resolver that applied the

AD bit is secured with cryptographic integrity protection. It is currently not clear how useful the AD bit will be in practice, but without a secured connection between the stub resolver and the validating resolver at the next hop, it is certainly useless.

A challenge facing deployment of DNSSEC is that non-validating stub resolvers often have no way of trusting their connection to the validating resolver at the next hop. Worse, for a client that wants to do its own validation, some ISPs block or otherwise alter traffic on the DNS port, so that it is not possible to receive an unaltered response that will pass DNSSEC validation. Other challenges include travelling users who are behind firewalls that modify DNS requests and responses, and only allow access to ports 80 and 443.

1.1. Known Problems with Intermediaries and Attackers

DNS intermediaries have been found to cause a number of tenacious and difficult-to-solve problems. An intermediary might change a query traversing it, might redirect the query to a different resolver, might alter a response coming back to a query, might try to parse a query or response and drop it if the intermediary doesn't understand it, or a combination of two or more of the above. [RFC5625] talks about some of the many ways that intermediaries have been known to make DNS resolution unstable (and, in some cases, unusable).

Attackers have shown great creativity in finding ways to spoof DNS responses. [RFC5452] describes some of these, but more are being discovered at a frequent pace. Spoofing does not affect DNS responses that are protected by DNSSEC that are sent to resolvers that validate; note, however, that non-validating resolvers, particularly non-validating stub resolvers, are quite common. Also note that this document only tries to prevent active attacks, and does not attempt to deal with denial-of-service (DoS) attacks on resolvers.

1.2. Proposed Solution

This document defines a method to tunnel in TLS [RFC5246] DNS traffic that is wrapped in HTTP [RFC2616]. Although these might at first seem like an ugly hack, they achieve the following goals:

- o Hiding DNS traffic from intermediaries that change DNS requests and/or responses on port 53

- o Cryptographic integrity protection for DNS responses sent to non-validating resolvers

- o Easy implementation in resolvers using existing toolkits and software

- o No changes to the DNS, HTTP, PKIX, or TLS protocols

- o Does not require that the responding resolvers run DNS protocols such as SIG(0) and TKEY that are open to trivial denial-of-service attacks.

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 RFC 2119 [RFC2119].

2. Description of the Protocol

This section defines the a protocol that first wraps encoded DNS in HTTP, then tunnels wrapped DNS in TLS.

The following steps are used by the querying resolver to create and send wrapped queries from one DNS resolver to a second resolver:

1. Start with the binary query that would have been sent to the second resolver.

2. Create a 128-bit nonce. Prepend it to the binary query from the previous step.

3. Encode the combined nonce-and-query with base64, as described in Section 4 of [RFC4648]. This results in a string of ASCII characters.

4. Create a URI with a path component consisting of "/.well-known/dnsreq/" prepended to the encoded nonce-and-query. (The "/.well-known/" prefix comes from [RFC5785]; the string "dnsreq" is registered with IANA later in this document.)

5. If a TLS session to the second resolver is already set up, that session is reused. If not, start a TLS session on port 443 with the second resolver. The querying resolver MUST verify that the TLS session is set up correctly before continuing.

6. Send an HTTP GET request with the URI from the previous step in the TLS tunnel. This request has an empty message body. Wait for the HTTP response.

Responding resolvers listen for TLS queries on TCP port 443. As appropriate, they leave the TLS session open to handle further

requests from the same client after the HTTP response is sent.

The following steps are used by the responding resolver to create and send wrapped responses to wrapped queries that come in from the HTTP GET request:

1. Strip off the "/.well-known/dnsreq/" preamble. If that preamble is not present in the request, then the request is for a different service and should be handled by that service.

2. The request to process is the remainder after stripping the preamble. Decode the request to process using base64. The first 128 bits of the decoded request is a nonce, and the rest is a DNS query.

3. Process the DNS request as one would any DNS request, as described in the DNS (and possibly DNSSEC) protocol documents. The result is a binary DNS response

4. Prepend the nonce to DNS response from the previous step and encode the resulting nonce-and-response using base64.

5. Respond to the HTTP request using a status code of 200, a media type of "text/plain", and the encoded DNS nonce-and-response in the HTTP response body. In order to help prevent overloading of any HTTP cache that may be between the HTTP server and client, the HTTP response SHOULD include a "Cache-Control" general-header field with the "no-store" directive, as described in RFC 2616.

The HTTP status code returned in the response is important to the querying resolover.

- Every DNS response MUST always be sent with an HTTP status code of 200 ("OK"). For example, even if the DNS response that is encoded is SERVFAIL or NXDOMAIN, the HTTP status code is still 200. Another way to look at this is if the HTTP service actually executes a query, the result will have an HTTP status code of 200. This design prevents the protocol from having to state one-by-one which DNS responses are "significant enough" errors to warrant an HTTP status code from the 4xx family.

- If the HTTP server knows that it cannot do the above processing, the HTTP response code MUST be 503 ("Service Unavailable"). An example of this would be if the HTTP server is up but that server knows that the DNS resolver service is down or cannot be reached from the HTTP server.

- o The HTTP server MAY send a status code in the 3xx family if the server has been moved to a different location; however, note that the client might not be able to follow this redirection, particularly if the location is given with a host name instead of an IP address.

- o Other HTTP-level error conditions (such as malformed request message, unexpected server-side problems, and so on) may yield other responses in the 4xx (client error) and 5xx (server error) range.

There are two possible types of HTTP responses, differentiated by the HTTP status code in the response. The originating DNS resolver acts differently for the two types of responses.

- o If the response has an HTTP status code of 200, the body of the HTTP response contains a single body part that is plain text. Decode the text using base64: the result of decoding is the 128-bit nonce followed by the DNS response. Validate that the nonce received is the same as the nonce that was sent; if not, ignore the response.

- o If the response has an HTTP status code of anything other than 200, the HTTP message body of the response can be ignored, and the only the HTTP status code (and possibly the HTTP status reason) is used by the querying resolver.

In order for the resolver to communicate over TLS to the second resolver using this protocol, the querying resolver needs a trust anchor to which the certificate proffered by the TLS server will chain. Without such a trust anchor, the TLS session will not start.

To reduce processing stress on the client and server, and to reduce DNS query times, TLS sessions SHOULD be long-lived. Further, for the same reason, TLS clients and servers SHOULD support TLS session resumption [RFC5077].

3. IANA Considerations

This document requests a new registration for a well-known URI, as defined in RFC 5785. The registration template is:

URI suffix: dnsreq

Change controller: IETF (when approved)

Specification document(s): This document

4. Security Considerations

The security considerations for the protocol are the same as those for TLS.

It is important to note that the nonce is only used to prevent cached HTTP responses from being served, not to prevent replay attacks in the inner HTTP protocol. If the inner HTTP queries and responses are cached, and the client reuses a nonce with the same DNS query, it could receive an out-of-date response instead of a fresh response. The use of fresh random nonces prevents this problem.

5. Acknowledgements

The ideas of wrapping DNS in HTTP-in-TLS have been discussed in many places by many people for a long time; the author of this document comes to the discussion quite late.

Andrew Sullivan did an early review of this document and contributed some of the text. Julian Reschke gave some good suggestions for the HTTP handling. Suggestions to ditch the HTTP-without-TLS protocol in an earlier draft came from many people.

6. References

6.1. Normative References

[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.

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

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

6.2. Informative References

[RFC2671] Vixie, P., "Extension Mechanisms for DNS (EDNS0)", RFC 2671, August 1999.

[RFC3205] Moore, K., "On the use of HTTP as a Substrate", BCP 56, RFC 3205, February 2002.

[RFC3225] Conrad, D., "Indicating Resolver Support of DNSSEC", RFC 3225, December 2001.

[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.

[RFC5077] Salowey, J., Zhou, H., Eronen, P., and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State", RFC 5077, January 2008.

[RFC5452] Hubert, A. and R. van Mook, "Measures for Making DNS More Resilient against Forged Answers", RFC 5452, January 2009.

[RFC5625] Bellis, R., "DNS Proxy Implementation Guidelines", BCP 152, RFC 5625, August 2009.

[RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, April 2010.

Appendix A. Appropriateness of Using HTTP as a Substrate

[RFC3205] describes best practices for using HTTP as a substrate, such as is done in this document. The following is an approximate checklist of how this protocol matches those practices. The section numbers are those from RFC 3205.

o 2.1: The complexity of HTTP is not significant here because standard clients and servers are used. No new headers or status codes are introduced, and normal HTTP paradigms like content

encoding are handled in their standard fashion.

- 2.2: The overhead is minimal for the size of DNS requests and responses.

- 2.3: The inner HTTP protocol does not expect to add any security to DNS. Authentication of the server in the full protocol is provided by TLS. Authentication of the client in the full protocol is not needed for DNS, although admission control to the server can be achieved in this protocol in the same way that it is for typical DNS, namely by IP address.

- 2.4: Compatibility with proxies, firewalls, and NATs is maximized by using TLS on port 443 in the protocol. Even if the HTTP in the inner protocol interacts with caching proxies, the nonce will cause caching proxies to never have repeated queries, and the SHOULD-level directive to include a "Cache-Control" general-header field with the "no-store" directive reduces the chance that HTTP-compliant proxies will be burdened by the high overhead of queries.

- 2.5, bullet 1: The payload size is only slightly increased, and DNS queries and response patterns can be similar to those seen on popular web sites.

- 2.5, bullet 2: This protocol is not meant to be used by web browsers.

- 2.5, bullet 3: Additional authentication is not needed in the inner HTTP protocol, and TLS adds the needed authentication for the full protocol.

- 2.5, bullet 4: Current HTTP status codes are fine for this protocol.

- 2.5, bullet 5: DNS servers do not normally support HTTP or other public-facing protocols.

- 3: The inner HTTP described in the protocol is not a substantially new service. It uses normal HTTP with the only significant restriction being on which response codes are used.

- 4: The http: and https: schemes are not expected to be used here. Instead, the HTTP requests that are wrapped in TLS are used directly.

- 5, bullet 1: An existing media type is used.

- 5, bullet 2: There is no need for multipart or messages when wrapping a DNS response.

- 5, bullet 3: Electronic mail is not a consideration here.

- 5, bullet 4: There is only one set of semantics for bodies in responses.

- 6: The GET method is used.

- 7: All existing client and server toolkits should be able to handle the few limitations in the protocol.

- 8: HTTP status codes are used as-is, and no new ones are created for the protocol.

Author's Address

Paul Hoffman
VPN Consortium

Email: paul.hoffman@vpnc.org