Hypertext Jeopardy Protocol (HTJP/1.0)

Abstract

   The Hypertext Jeopardy Protocol (HTJP) inverts the request/response
   semantics of the Hypertext Transfer Protocol (HTTP). Using
   conventional HTTP, one connects to a server, asks a question, and
   expects a correct answer. Using HTJP, one connects to a server,
   sends an answer, and expects a correct question. This document
   specifies the semantics of HTJP.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

   The Hypertext Jeopardy Protocol (HTJP) 1.0 is a stateless
   application-level response/request protocol that functions as the
   semantic inverse of the Hypertext Transfer Protocol (HTTP) 1.1 .

   It can roughly be specified in relation to HTTP by the following
   rules:

   o Where an HTTP client would send an HTTP request message, an HTJP
      client would send an HTTP response message.

   o Where an HTTP server would send an HTTP response message, an HTJP
      server would send an HTTP request message.

   o The HTTP request sent as an HTJP response should be an HTTP
      request that (if sent to the appropriate HTTP server) would elicit
      the HTTP response sent in the HTJP request.

   HTJP is compatible with the HTTP/1.1 specification, at least in
   spirit, if not in letter.

HTJP has novel applications in all the following areas:

o Generative automated testing of HTTP implementations and HTTP-
   based applications.

o Monitoring of HTTP-based applications in production.

o Forensic and diagnostic reconstruction of HTTP requests from HTTP
   response logs.

o Discovery of first-party and third-party security vulnerabilities.

2. Conventions Used in This Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
"SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and
"OPTIONAL" in this document are to be interpreted as described in
BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all
capitals, as shown here.

3. Comparison with HTTP

It is a little-known fact that HTTP/1.1 already defines itself to be
its own inverse mode of operation. Section 3.1 of RFC 7230
[RFC7230], which describes the start line of the HTTP message format,
states:

    In theory, a client could receive requests and a server could
    receive responses, distinguishing them by their different start-
    line formats, but, in practice, servers are implemented to only
    expect a request [...] and clients are implemented to only expect
    a response.

It is only convention that HTTP clients send HTTP requests and that
HTTP servers send HTTP responses. Therefore, HTJP is just HTTP with
some alternative conventions. It is not a distinct protocol.
Furthermore, since all messages in HTJP are indistinguishable from
HTTP messages, an HTJP peer would have no way of identifying itself
explicitly as using HTJP rather than HTTP.

Therefore, we describe HTJP as a "pseudo-protocol" in order to
distinguish clients, servers, and conversations that are using the
HTTP conventions laid out in this document from those that use
conventions that are more commonly associated with HTTP.

4. Response and Request Semantics

   An HTJP request MUST be an HTTP response message. An HTJP response
   message MUST be an HTTP request message that, if issued to the
   appropriate HTTP server, would elicit the HTTP response specified by
   the HTJP request being replied to.

   As described in Section 3, HTJP is unconventional but valid HTTP, and
   so the entirety of the HTTP specification (as defined in [RFC7230],
   [RFC7231], [RFC7232], [RFC7233], [RFC7234], and [RFC7235]) MUST be
   respected when doing so is consistent with HTJP's unique semantics.

   The following example illustrates a typical message exchange for an
   HTJP request concerning the same hypothetical server from Section 2.1
   of RFC 7230 [RFC7230].

   Client request:

      HTTP/1.1 200 OK
      Date: Mon, 27 Jul 2009 12:28:53 GMT
      Server: Apache
      Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
      ETag: "34aa387-d-1568eb00"
      Accept-Ranges: bytes
      Content-Length: 51
      Vary: Accept-Encoding
      Content-Type: text/plain

      Hello World! My payload includes a trailing CRLF.

   Server response:

      GET /hello.txt HTTP/1.1
      Host: www.example.com

4.1. Applicability of Postel's Robustness Principle

   Implementations of HTJP SHOULD respect Postel's Robustness Principle
   [IAB-PROTOCOL-MAINTENANCE].

   Applied to HTJP, Postel's Robustness Principle implies that, given
   the choice of multiple valid HTJP responses for an HTJP request, one
   SHOULD prefer a response that is more adherent to the HTTP standard
   or uses fewer HTTP features. For example, sometimes a User-Agent
   header has no bearing on the HTTP response from an HTTP server. On
   seeing such a response in an HTJP request, an HTJP server could
   validly respond with a practically unlimited number of permutations
   on the User-Agent header value. However, it SHOULD prefer to respond

with an HTTP request that has no User-Agent header whatsoever, in
keeping with Postel's Robustness Principle.

The same consideration applies when encountering an HTJP request for
which there are both valid and "pseudo-valid" (Section 4.4) HTJP
responses available.

4.2. Identifying the Server Associated with an HTJP Response

It may be of interest to a user of HTJP to try issuing an HTJP
response as an HTTP request to the appropriate server. This brings
up the issue of correctly identifying the host to which the HTJP
response should be sent. Much of the time this will be able to be
determined from the Host header field of the HTJP response. The Host
header is required by conformant HTTP/1.1 requests. In the case that
the Host header is not present (for example, if the HTJP response is
an HTTP/1.0 request rather than HTTP/1.1), an HTJP response MAY use
the absolute URI form in the HTTP request line, to add clarity about
the target host if it would be validly accepted by the server. This
specific example is complicated by the fact that prior to HTTP/1.1 it
was not required that implementations accept the absolute URI form.
For this reason, it is also possible to phrase the HTJP response as
an HTTP request to a Forward Proxy server, which would have accepted,
indeed needed, the absolute URI request line prior to and after
HTTP/1.1. As a last resort, it may be possible to heuristically
derive the target host of an HTJP response from the HTJP request; for
example, the HTJP request may have absolute references to other HTTP
resources that seem to come from the same host.

4.3. Temporal Considerations

When an HTJP response is issued, there is no guarantee that, by the
time the response is received by an HTJP client, the HTTP server that
is associated with said response will still reply with it. Providing
any guarantee about "when" an HTTP server would reply with said
response is obviously a theoretically unsolvable problem and
therefore outside the scope of this HTJP specification. It is only
required that the HTJP response be correct at some point in the range
of the 32-bit Unix Timestamp; see "Seconds Since the Epoch"
(Section 4.16) of Unix General Concepts [UNIX-General-Concepts].

HTJP servers that are responding with an HTTP request for a volatile
resource, and with high confidence in the time range at which the
resource would be in the state described by the HTJP request, MAY add
a Date header to the HTJP response. This is in conformance with the
ability for HTTP requests to carry a Date header; see Section 7.1.1.2
of [RFC7231].

HTJP clients can try to demand more temporal certainty by adding a
Date header to their HTTP response, embedding criteria for the time
of the HTTP response in the HTTP response itself. Of course, the
client might still only receive that exact HTTP response if it
manages to deliver the HTTP request at the precise time of the
previously requested Date header, and even then it is still not
guaranteed due to HTTP caching et cetera.

4.4. Pseudo-Valid HTJP Messages

In the wild, HTTP clients and servers have been known to occasionally
exchange HTTP messages that are not conformant to any HTTP
specification. For this reason, we will identify a class of messages
that are, on the one hand, invalid HTTP messages, yet at the same
time, correctly answerable HTJP requests or correct answers to an
HTJP request, as "pseudo-valid" HTJP messages.

Consider, for example, an HTTP server that erroneously reports a
Content-Length header field of zero when sending an HTTP payload of
non-zero length. Despite this HTTP message violating the HTTP
specification, it is possible for an HTJP server to receive such a
message and correctly respond to it, satisfying the HTJP semantics in
doing so.

This applies to both HTJP requests and HTJP responses. There may be
times when the only valid HTJP response is an invalid HTTP request.
When there are both valid and invalid HTTP requests that could
satisfy the HTJP request, Postel's Robustness Principle SHOULD be
applied, as described in Section 4.1.

4.5. HTTP Responses That Are Not Requestable

Given that an HTJP response MUST be an HTTP request, and that HTTP
requests do not have a status field (such as a status code), there is
no way for an HTJP response to signal a failure in response to an
HTJP request, via a status code or otherwise. The correct HTJP
response to an HTJP request when a server cannot determine an HTTP
request that elicits the HTTP response is to not respond at all. The
HTJP responder MAY close the connection; however, the HTJP requester
MUST NOT interpret the closing of the connection as a response. This
can have issues when HTJP servers are hosted behind non-HTJP-aware
HTTP proxies, as the proxy may inject a 5xx HTTP response, which
could be misinterpreted as an HTJP response. See more on proxies in
Section 5.

5. Caches and Proxies

   Despite being valid HTTP traffic, support for caching and proxying of
   HTJP traffic is unfortunately not widespread. It is estimated that
   currently approximately 0.0% of all HTTP-aware intermediaries
   gracefully handle HTJP traffic. For this reason, it is currently
   RECOMMENDED that HTJP should not be deployed behind an HTTP proxying
   or caching layer. Support will likely increase proportionally to
   HTJP's popularity in production.

6. IANA Considerations

   In order that implementations of this specification can operate
   correctly, and in accordance with [BCP90], the following new HTTP
   header field would need to be registered in the "Message Headers"
   registry maintained at
   <https://www.iana.org/assignments/message-headers/>. To find out the
   details of this registration, an application should send an HTJP
   request to a suitable server specifying the Anti-HTJP-Nonce value
   that the server would have supplied had the IANA registration been
   made.

   +--------------------+----------+---------------+---------------+
   | Header Field Name | Protocol | Status | Reference |
   +--------------------+----------+---------------+---------------+
   | Anti-HTJP-Nonce | http | informational | Section 7.1.1 |
   +--------------------+----------+---------------+---------------+

7. Security Considerations

7.1. Securing HTTP against HTJP

   An incomplete implementation of HTJP is inadvisable from a security
   perspective. A complete implementation of HTJP may have interesting
   security features that are worthy of detailed examination. Due to
   its semantics, the issuing of a successfully authorized HTTP response
   to an HTJP server will result in a reply containing the HTTP request
   that elicits said response, including any credentials, tokens,
   cookies, or other authorization materials that were required to
   elicit that response.

As an example:

Client request:

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Content-Length: 61
Content-Type: text/plain

Some predictable information accessed using authorization.
```

Server response:
(line breaks in the Authorization header are for RFC formatting)

```
GET /information.txt HTTP/1.1
Host: authorised-usage-service.example.com
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
    eyJzdWIiOiJodGpwIiwibmFtZSI6IkV2ZXJ5b25lIiwiaWF0IjowfQ.
    JOL-kIObgTI0MzFfm1yVFFkIo1xf7DZGjY_oedRBZW0
```

Given that we cannot prevent anyone from attempting to implement HTJP, it is RECOMMENDED to consider how HTJP impacts security when using HTTP.

Note that it was only possible to query for the credentialed HTTP request because the response to the authorized request was predictable. HTTP servers could mitigate this vulnerability exposed by HTJP by only serving a response that is at least as secret as the credentials themselves in response to an authorized request.

7.1.1. Anti-HTJP-Nonce Header

A generic solution to this problem is to use an "Anti-HTJP-Nonce" HTTP header in HTTP responses. The value of an "Anti-HTJP-Nonce" header SHOULD be a cryptographically secure random number in any encoding that is valid for an HTTP header value. The length of this number SHOULD be determined by the producer of the HTTP response, accounting for their method of random number generation and their threat model.

7.2. HTJPS

HTJP, being just HTTP, has most of the same security concerns and features as HTTP itself. For example, the use of HTJP over an encrypted connection, such as TLS 1.3 [RFC8446], similar to HTTP Secure (HTTPS), is referred to as HTJP Secure (HTJPS). However, it is important to note that, unlike with HTTPS, it is not expected that the hostname you are securely communicating with is the same hostname

as featured in the Host headers or absolute URIs of the HTJP messages
themselves, as HTJP messages are typically referring to other HTTP
hosts. This excludes the case of a server that supports both
conventional HTTP and HTJP, where it is possible to make HTJP
requests securely to the same host that is also the subject of the
HTJP requests being made.

8. References

8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
           Requirement Levels", BCP 14, RFC 2119,
           DOI 10.17487/RFC2119, March 1997,
           <https://www.rfc-editor.org/info/rfc2119>.

[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,
           <https://www.rfc-editor.org/info/rfc7230>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer
           Protocol (HTTP/1.1): Semantics and Content", RFC 7231,
           DOI 10.17487/RFC7231, June 2014,
           <https://www.rfc-editor.org/info/rfc7231>.

[RFC7232] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer
           Protocol (HTTP/1.1): Conditional Requests", RFC 7232,
           DOI 10.17487/RFC7232, June 2014,
           <https://www.rfc-editor.org/info/rfc7232>.

[RFC7233] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed.,
           "Hypertext Transfer Protocol (HTTP/1.1): Range Requests",
           RFC 7233, DOI 10.17487/RFC7233, June 2014,
           <https://www.rfc-editor.org/info/rfc7233>.

[RFC7234] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke,
           Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching",
           RFC 7234, DOI 10.17487/RFC7234, June 2014,
           <https://www.rfc-editor.org/info/rfc7234>.

[RFC7235] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer
           Protocol (HTTP/1.1): Authentication", RFC 7235,
           DOI 10.17487/RFC7235, June 2014,
           <https://www.rfc-editor.org/info/rfc7235>.

   [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC
              2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174,
              May 2017, <https://www.rfc-editor.org/info/rfc8174>.

   [UNIX-General-Concepts]
              "General Concepts", Chapter 4 of "The Open Group Base
              Specifications, Issue 7", 2018 edition, IEEE
              Std 1003.1-2017, 2018, <http://pubs.opengroup.org/
              onlinepubs/9699919799/basedefs/V1_chap04.html>.

8.2. Informative References

   [BCP90]   Klyne, G., Nottingham, M., and J. Mogul, "Registration
              Procedures for Message Header Fields", BCP 90, RFC 3864,
              September 2004, <https://www.rfc-editor.org/info/bcp90>.

   [IAB-PROTOCOL-MAINTENANCE]
              Thomson, M., "The Harmful Consequences of the Robustness
              Principle", Work in Progress, draft-iab-protocol-
              maintenance-02, March 2019.

   [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol
              Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018,
              <https://www.rfc-editor.org/info/rfc8446>.

Appendix A. Hypertext Double Jeopardy Protocol

   Also worth mentioning, in case one encounters it in the wild, is the
   Hypertext Double Jeopardy Protocol (HTJ2P). The Hypertext Double
   Jeopardy Protocol 1.0 is a stateless application-level request/
   response protocol that functions as the inverse of the Hypertext
   Jeopardy Protocol (HTJP) 1.0 .

   An HTJ2P response MUST be an HTTP response which would be issued for
   an HTTP request delivered as the HTJ2P request. Implementations of
   HTJ2P have groundbreaking potential in the fields of HTTP caching,
   and in the implementation of HTJP.

Acknowledgements

   The author thanks Alex Trebek for his distinguished contributions to
   culture and society. The author thanks Peter Phillips for the
   suggestion of the Anti-HTJP-Nonce header. The author also wishes to
   thank anyone who has ever built a tool or a technology that made
   people ask "Why?".

Author's Address

   Edmund Fokschaner

   Email: edfokschaner@gmail.com