Network Working Group
Request for Comments: 4236
Category: Standards Track

A. Rousskov
The Measurement Factory
M. Stecher
CyberGuard Corporation

November 2005

HTTP Adaptation with Open Pluggable Edge Services (OPES)

Status of This Memo

Copyright Notice

Abstract

Open Pluggable Edge Services (OPES) framework documents several application-agnostic mechanisms such as OPES tracing, OPES bypass, and OPES callout protocol. This document extends those generic mechanisms for Hypertext Transfer Protocol (HTTP) adaptation. Together, application-agnostic OPES documents and this HTTP profile constitute a complete specification for HTTP adaptation with OPES.

Table of Contents

1. Scope

The Open Pluggable Edge Services (OPES) framework documents several application-agnostic mechanisms such as OPES processor and endpoints communications [RFC3897] or OPES callout protocol [RFC4037]. This document extends those generic mechanisms for adaptation of a specific application protocol, HTTP [RFC2616]. Together, application-agnostic OPES documents and this HTTP profile constitute a complete specification for HTTP adaptation with OPES.

The primary sections of this document specify HTTP-specific extensions for the corresponding application-agnostic mechanisms documented elsewhere.

2. OPES Document Map

This document belongs to a large set of OPES specifications produced by the IETF OPES Working Group. Familiarity with the overall OPES approach and typical scenarios is often essential when trying to comprehend isolated OPES documents. This section provides an index of OPES documents to assist the reader with finding "missing" information.

- The document on "OPES Use Cases and Deployment Scenarios" [RFC3752] describes a set of services and applications that are considered in scope for OPES and have been used as a motivation and guidance in designing the OPES architecture.

- The OPES architecture and common terminology are described in "An Architecture for Open Pluggable Edge Services (OPES)" [RFC3835].

- "Policy, Authorization and Enforcement Requirements of OPES" [RFC3838] outlines requirements and assumptions on the policy framework, without specifying concrete authorization and enforcement methods.

- "Security Threats and Risks for OPES" [RFC3837] provides OPES risk analysis, without recommending specific solutions.

- "OPES Treatment of IAB Considerations" [RFC3914] addresses all architecture-level considerations expressed by the IETF Internet Architecture Board (IAB) when the OPES WG was chartered.

- At the core of the OPES architecture are the OPES processor and the callout server, two network elements that communicate with each other via an OPES Callout Protocol (OCP). The requirements for such protocol are discussed in "Requirements for OPES Callout Protocols" [RFC3836].

- "OPES Callout Protocol Core" [RFC4037] specifies an application agnostic protocol core to be used for the communication between OPES processor and callout server.

- "OPES entities and end points communications" [RFC3897] specifies generic tracing and bypass mechanisms for OPES.

- The OCP Core and Communications documents are independent from the application protocol being adapted by OPES entities. Their

  generic mechanisms have to be complemented by application-specific profiles. This document, HTTP adaptation with OPES, is such an application profile for HTTP. It specifies how application-agnostic OPES mechanisms are to be used and augmented in order to support adaptation of HTTP messages.

- Finally, "P: Message Processing Language" [rules-p] defines a language for specifying what OPES adaptations (e.g., translation) must be applied to what application messages (e.g., e-mail from bob@example.com). P language is meant for configuring application proxies (OPES processors).

3. Callout Protocol

This section documents the HTTP profile for the OPES Callout Protocol (OCP) Core [RFC4037]. Familiarity with OCP Core is required to understand the HTTP profile. This section uses OCP Core conventions, terminology, and mechanisms.

OPES processor communicates its desire to adapt HTTP messages via a Negotiation Offer (NO) message with HTTP-specific feature identifiers documented in Section 3.2. HTTP-specific OCP optimization mechanisms can be negotiated at the same time. A callout server that supports adaptation of HTTP messages has a chance to negotiate what HTTP message parts will participate in adaptation, including negotiation of HTTP request parts as metadata for HTTP response adaptation. Negotiable HTTP message parts are documented in Section 3.1.

HTTP profile introduces a new parameter for the Application Message Start (AMS) message to communicate known HTTP message length (HTTP headers often do not convey length information reliably or at all). This parameter is documented in Section 3.3. Section 3.4 documents a mechanism to report HTTP message parts with Data Use Mine (DUM) messages.

The remaining OCP sections document various OCP marshaling corner cases such as handling of HTTP transfer encodings and 100 Continue responses.

3.1. Application Message Parts

An HTTP message may have several well-known parts: headers, body, and trailers. HTTP OPES processors are likely to have information about HTTP message parts because they have to isolate and interpret HTTP headers and find HTTP message boundaries. Callout servers may either not care about certain parts or may benefit from reusing HTTP OPES processor work on isolating and categorizing interesting parts.

The following is the declaration of am-part (application message part) type using OCP Core Protocol Element Type Declaration Mnemonic (PETDM):

```
am-part:  extends atom;
am-parts: extends list of am-part;
```

Figure 1

The following six "am-part" atoms are valid values:

request-header: The start-line of an HTTP request message, all request message headers, and the CRLF separator at the end of HTTP headers (compare with section 4.1 of [RFC2616]).

request-body: The message body of an HTTP request message as defined in section 4.3 of [RFC2616] but not including the trailer.

request-trailer: The entity headers of the trailer of an HTTP request message in chunked transfer encoding. This part follows the same syntax as the trailer defined in section 3.6.1 of [RFC2616].

response-header: The start-line of an HTTP response message, all response message headers, and the CRLF separator at the end of HTTP headers (compare with section 4.1 of [RFC2616]).

response-body: The message body of an HTTP response message as defined in section 4.3 of [RFC2616] but not including the trailer.

response-trailer: The entity headers of the trailer of an HTTP response message in chunked transfer encoding. This part follows the same syntax as the trailer defined in section 3.6.1 of [RFC2616].

3.2. Application Profile Features

This document defines two HTTP profiles for OCP: request and response profiles. These two profiles are described below. Each profile has a unique feature identifier, a list of original application message parts, and a list of adapted application message parts:

profile ID: http://www.iana.org/assignments/opes/ocp/http/request

- original request parts: request-header, request-body, request-trailer
- adapted request parts: request-header, request-body, request-trailer
- adapted response parts: response-header, response-body, response-trailer

profile ID: http://www.iana.org/assignments/opes/ocp/http/response

- original transaction parts: request-header (aux), request-body (aux), request-trailer (aux), response-header, response-body, response-trailer
- adapted response parts: response-header, response-body, response-trailer

The request profile contains two variants of adapted part lists: HTTP request parts and HTTP response parts. Parts marked with an "(aux)" suffix are auxiliary parts that can only be used if explicitly negotiated for a profile. See Section 3.2.1 for specific rules governing negotiation and use of am-parts.

The scope of a negotiated profile is the OCP connection (default) or the service group specified via the SG parameter.

3.2.1. Profile Parts

An OCP agent MUST send application message parts in the order implied by the profile parts lists above. An OCP agent receiving an out-of-order part MAY terminate the transaction with an error.

An OPES processor MUST NOT send parts that are not listed as "original" in the negotiated profile. A callout server MUST NOT send parts that are not listed as "adapted" in the negotiated profile. An OCP agent receiving an not-listed part MUST terminate the transaction with an error. The informal rationale for the last requirement is to reduce the number of subtle interoperability problems where an agent

thinks that the parts it is sending are understood/used by the other agent when, in fact, they are being ignored or skipped because they are not expected.

Some HTTP messages lack certain parts. For example, many HTTP requests do not have bodies, and most HTTP messages do not have trailers. An OCP agent MUST NOT send (i.e., must skip) absent application message parts.

An OCP agent MUST send present non-auxiliary parts and it MUST send those present auxiliary parts that were negotiated via the Aux-Parts (Section 3.2.3) parameter. OCP agents MUST NOT send auxiliary parts that were not negotiated via the Aux-Parts (Section 3.2.3) parameter.

An OCP agent receiving a message part in violation of the above requirements MAY terminate the corresponding transaction with an error.

By design, original parts not included in the adapted parts list cannot be adapted. In other words, a callout service can only adapt parts in the adapted parts list even though it may have access to other parts.

In the request profile, the callout server MUST send either adapted

request parts or adapted response parts. An OPES processor receiving adapted flow with application message parts from both lists (in violation of the previous rule) MUST terminate the OCP transaction with an error. Informally, the callout server sends adapted response parts to "short-circuit" the HTTP transaction, forcing the OPES processor to return an HTTP response without forwarding an adapted HTTP request. This short-circuiting is useful for responding, for example, to an HTTP request that the callout service defines as forbidden.

Unless explicitly configured to do otherwise, an OPES processor MUST offer all non-auxiliary original parts in Negotiation Offer (NO) messages. See Section 3.5 for this rule rationale and examples of harmful side-effects from selective adaptation.

3.2.2. Profile Structure

An HTTP application profile feature extends semantics of the feature type of OCP Core while adding the following named parameters to that type:

- Aux-Parts (Section 3.2.3)
- Pause-At-Body (Section 3.2.4)
- Stop-Receiving-Body (Section 3.2.5)
- Preservation-Interest-Body (Section 3.2.6)
- Content-Encodings (Section 3.2.7)

The definition of the HTTP profile feature structure using PETDM follows:

```
HTTP-Profile: extends Feature with {
    [Aux-Parts: am-parts];
    [Pause-At-Body: size];
    [Stop-Receiving-Body: size];
    [Preservation-Interest-Body: size];
    [Content-Encodings: codings];
};
```

Figure 2

An HTTP profile structure can be used in feature lists of Negotiation Offer (NO) messages and as an anonymous parameter of a Negotiation Response (NR) message. All profile parameters apply to any OCP transaction within profile scope.

3.2.3. Aux-Parts

The Aux-Parts parameter of an HTTP response profile can be used to negotiate the inclusion of auxiliary application message parts into the original data flow. The parameter is a possibly empty list of am-part tokens. An OPES processor MAY send an Aux-Parts parameter to advertise availability of auxiliary application message parts. A callout server MAY respond with a possibly empty subset of the parts it needs. The callout server response defines the subset of successfully negotiated auxiliary message parts.

When receiving a Negotiation Offer (NO) message, the callout server MUST ignore any non-auxiliary part listed in the Aux-Parts parameter. When sending a Negotiation Response (NR) message, the callout server

MUST NOT select any application message part that was not explicitly listed in the negotiation offer. In case of a violation of the last rule, the OPES processor MUST terminate the transaction.

An OPES processor MUST send each negotiated auxiliary part to the callout server, unless the part is absent.

Example:

```
Aux-Parts: (request-header,request-body)
```

Figure 3

3.2.4. Pause-At-Body

A callout server MAY use the Pause-At-Body parameter to request a pause in original application message body transmission before original dataflow starts. The parameter's value is of type "offset". The parameter specifies the start of the non-auxiliary application message body suffix that the sender is temporarily not interested in seeing.

```
[headers][ body prefix | body suffix ][trailer]
<-- ? --><-- offset  --><-- ? ---------------->
<-- equiv. DWP offset ->
```

Figure 4

When an OPES processor receives a Pause-At-Body parameter, it MUST behave as if it has received a Want Data Paused (DWP) message with the corresponding org-offset. Note that the latter offset is different from the Pause-At-Body offset and is unknown until the size of the HTTP message headers is known.

For example, if the Pause-At-Body value is zero, the OPES processor should send a Paused My Data (DPM) message just before it sends the first Data Use Mine (DUM) message with the response-body part in the HTTP response profile. If the Pause-At-Body value is 300, the OPES processor should send a DPM message after transmitting 300 OCTETs for that application message part.

Example:

```
Pause-At-Body: 0
```

Figure 5

3.2.5. Stop-Receiving-Body

A callout server MAY use the Stop-Receiving-Body parameter to imply a Want Stop Receiving Data (DWSR) message behavior before the original dataflow starts. The parameter's value is of type "offset". The parameter specifies an offset into the original, non-auxiliary message body part (request-body in request profile and response-body in response profile).

A callout service MAY send a Stop-Receiving-Body parameter with its negotiation response if there is a fixed offset into the message body for all transactions of a profile for which a Want Stop Receiving Data (DWSR) message would be sent. An OPES processor MUST behave as if it has received a DWSR message with the corresponding offset. Note that the latter offset is different from the Stop-Receiving-Body offset and is unknown until the size of the HTTP message headers is known.

For example, if the Stop-Receiving-Body value is zero in an HTTP response profile, the OPES processor should send an Application Message End (AME) message with result code 206 immediately after sending the response-header message part and before starting with the response-body message part.

Example:

```
Stop-Receiving-Body: 0
```

Figure 6

3.2.6. Preservation-Interest-Body

The Preservation-Interest-Body parameter can be used to optimize data preservation at the OPES processor. The parameter's value is of type "size" and denominates a prefix size of the original, non-auxiliary message body part (request-body in HTTP request profile and response-body in response profile).

A callout service MAY send a Preservation-Interest-Body parameter with its negotiation response if there is a fixed-size prefix of the application message body for which a Data Preservation Interest (DPI) message would be sent. An OPES processor MUST behave as if it receives a DPI message with org-offset zero and org-size equal to the value of the Preservation-Interest-Body parameter.

For example, if the Preservation-Interest-Body value is zero in an HTTP response profile, the callout server must not send any Data Use Yours (DUY) message for the response-body part; the OPES processor may use this information to optimize its data preservation behavior even before it makes the decision to preserve data.

Example:

```
Preservation-Interest-Body: 0
```

Figure 7

3.2.7. Content-Encodings

A callout server MAY send a Content-Encodings list to indicate its preferences in content encodings. Encodings listed first are preferred to other encodings. An OPES processor MAY use any content encoding when sending application messages to a callout server.

The list of preferred content encodings does not imply lack of support for other encodings. The OPES processor MUST NOT bypass a service just because the actual content encoding does not match the service's preferences.

If an OCP agent receives an application message that it cannot handle due to specific content encoding, the usual transaction termination rules apply.

```
content-coding: extends atom;
content-codings: extends list of content-coding;
```

Example:

```
Content-Encodings: (gzip)
```

Figure 8

The semantics of content-coding is defined in section 3.5 of [RFC2616].

3.2.8. Profile Negotiation Example

```
Example:
  P: NO ({"54:http://www.iana.org/assignments/opes/ocp/http/response"
     Aux-Parts: (request-header,request-body)
     })
     SG: 5
     ;
  S: NR {"54:http://www.iana.org/assignments/opes/ocp/http/response"
     Aux-Parts: (request-header)
     Pause-At-Body: 30
     Preservation-Interest-Body: 0
     Content-Encodings: (gzip)
     }
     SG: 5
     ;
```

Figure 9

This example shows a negotiation offer made by an OPES processor for a service group (id 5) that has already been created; the callout server sends an adequate negotiation response.

The OPES processor offers one profile feature for HTTP response messages. Besides the standard message parts, the OPES processor is able to add the header and body of the original HTTP request as auxiliary message parts.

The callout server requests the auxiliary request-header part, but is not interested in receiving the request-body part.

The OPES processor sends at most the following message parts, in the specified order, for all transactions in service group 5: request-header, response-header, response-body, response-trailer. Note that the request-body part is not included (because it is an auxiliary part that was not explicitly requested). Some of the response parts may not be sent if the original message lacks them.

The callout server indicates through the Preservation-Interest-Body parameter with size zero that it will not send any DUY messages. The OPES processor may therefore preserve no preservation for any transaction of this profile.

By sending a Pause-At-Body value of 30, the callout server requests a data pause. The OPES processor sends a Paused My Data (DPM) message immediately after sending at least 30 OCTETs of the response-body part. Thereafter, the OPES processor waits for a Want More Data (DWM) message from the callout service.

3.3. Application Message Start Message

A new named parameter for Application Message Start (AMS) messages is introduced.

AM-EL: size

Figure 10

AM-EL value is the size of the request-body part in the HTTP request profile, and is the size of the response-body part in the HTTP response profile, before any transfer codings have been applied (or after all transfer codings have been removed). This definition is consistent with the HTTP entity length definition.

An OCP agent that knows the exact length of the HTTP message entity (see Section 7.2.2 "Entity Length" in [RFC2616]) at the time it sends the AMS message, SHOULD announce this length using the AM-EL named parameter of an AMS message. If the exact entity length is not known, an OCP agent MUST NOT send an AM-EL parameter. Relaying correct entity length can have significant performance advantages for the recipient, and implementations are strongly encouraged to relay known entity lengths. Similarly, relaying incorrect entity length can have drastic correctness consequences for the recipient, and implementations are urged to exercise great care when relaying entity length.

An OPES processor receiving an AM-EL parameter SHOULD use the parameter’s value in a Content-Length HTTP entity header when constructing an HTTP message, provided a Content-Length HTTP entity header is allowed for the given application message by HTTP (see Section 3.8.1).

3.4. DUM Message

A new named parameter for Data Use Mine (DUM) messages is introduced.

AM-Part: am-part

Figure 11

An OCP agent MUST send an AM-Part parameter with every DUM message that is a part of an OCP transaction with an HTTP profile. The AM-Part parameter value is a single am-part token. As implied by the syntax, a DUM message can only contain data of a single application message part. One message part can be fragmented into any number of DUM messages with the same AM-Part parameter.

The following example shows three DUM messages containing an abridged HTTP response message. The response-body part is fragmented and sent within two DUM messages.

Example:

```
    P: DUM 88 1 0
       Kept: 0
       AM-Part: response-header

       64:HTTP/1.1 200 OK
       Content-Type: text/html
       Content-Length: 51

       ;
    P: DUM 88 1 64
       Kept: 64
       AM-Part: response-body

       19:<html><body>This is
       ;
    P: DUM 88 1 83
       Kept: 83
       AM-Part: response-body

       32: a simple message.</body></html>
       ;
```

Figure 12

3.5. Selective Adaptation

The HTTP profile for OCP applies to all HTTP messages. That scope includes HTTP messages such as 1xx (Informational) responses, POST, CONNECT, and OPTIONS requests, as well as responses with extension status codes and requests with extension methods. Unless specifically configured to do otherwise, an OPES processor MUST forward all HTTP messages for adaptation at callout servers. OPES bypass instructions, configured HTTP message handling rules, and OCP-negotiation with a callout server are all examples of an acceptable "specific configuration" that provides an exception to this rule.

While it may seem useless to attempt to adapt "control" messages such as a 100 (Continue) response, skipping such messages by default may lead to serious security flaws and interoperability problems. For example, sensitive company information might be relayed via a

carefully crafted 100 Continue response; or a malicious CONNECT request may not get logged if OPES processor does not forward these messages to a callout service that is supposed to handle them.

By design, OPES processor implementation cannot unilaterally decide that an HTTP message is not worth adapting. It needs a callout server opinion, a configuration setting, or another external information to make the decision.

3.6. Hop-by-hop Headers

HTTP defines several hop-by-hop headers (e.g., Connection) and allows for extension headers to be specified as hop-by-hop ones (via the Connection header mechanism). Depending on the environment and configuration, an OPES processor MAY forward hop-by-hop headers to callout servers and MAY use hop-by-hop headers returned by callout servers to build an HTTP message for the next application hop. However, see Section 3.7 for requirements specific to the Transfer-Encoding header.

For example, a logging or statistics collection service may want to see hop-by-hop headers sent by the previous application hop to the OPES processor and/or hop-by-hop headers sent by the OPES processor to the next application hop. Another service may actually handle HTTP logic of removing and adding hop-by-hop headers. Many services will ignore hop-by-hop headers. This specification does not define a mechanism for distinguishing these use cases.

3.7. Transfer Encodings

HTTP messages may use transfer encodings, a hop-by-hop encoding feature of HTTP. Adaptations that use HTTP transfer encodings have to be explicitly negotiated. This specification does not document such negotiations. In the absence of explicit transfer-encoding negotiations, an OCP agent MUST NOT send transfer-encoded application message bodies.

Informally, the above rule means that the agent or its environment have to make sure that all transfer encodings are stripped from an HTTP message body before it enters OCP scope. An agent MUST terminate the OCP transaction if it has to send an application message body but cannot remove all transfer encodings. Violations of these rules lead to interoperability problems.

If an OCP agent receives transfer-encoded application data in violation of the above requirement, the agent MAY terminate the corresponding OCP transaction.

An OPES processor removing transfer encodings MUST remove the Transfer-Encoding header before sending the header part to the callout service. A callout server receiving a Transfer-Encoding header MAY assume that original application data is still transfer-encoded (and terminate the transaction). The OPES processor MUST send a correct Transfer-Encoding header to the next HTTP recipient, independent of what header (if any) the callout server returned.

Logging and wiretapping are the examples where negotiating acceptable transfer encodings may be worthwhile. While a callout server may not be able to strip an encoding, it may still want to log the entire message "as is". In most cases, however, the callout server would not be able to meaningfully handle unknown transfer encodings.

3.8. HTTP Header Correctness

When communicating with HTTP applications, OPES processors MUST ensure correctness of all computable HTTP headers documented in specifications that the processors intend to be compliant with. A computable header is defined as a header whose value can be computed based on the message body alone. For example, the correctness of Content-Length and Content-MD5 headers has to be ensured by processors claiming compliance with HTTP/1.1 ([RFC2616]).

Informally and by default, the OPES processor has to validate and eventually recalculate, add, or remove computable HTTP headers in order to build a compliant HTTP message from an adapted application message returned by the callout server. If a particular OPES processor trusts certain HTTP headers that a callout service sends, it can use those headers "as is".

An OPES processor MAY forward a partially adapted HTTP message from a callout server to the next callout server, without verifying HTTP header correctness. Consequently, a callout service cannot assume that the HTTP headers it receives are correct or final from an HTTP point of view.

The following subsections present guidelines for the recalculation of some HTTP headers.

3.8.1. Message Size Recalculation

By default, an OCP agent MUST NOT trust the Content-Length header that is sent within an HTTP header message part. The message length could be modified by a callout service without adaptation of the HTTP message headers.

Before sending the HTTP message to the HTTP peer, the OPES processor has to ensure correctness of the message length indication according to section 4.4 of [RFC2616].

Besides ensuring HTTP message correctness, good OPES processors set up the message to optimize performance, including minimizing delivery latency. Specifically, indicating the end of a message by closing the HTTP connection ought to be the last resort:

- If the callout server sends an AM-EL parameter with its AMS message, the OPES processor SHOULD use this value to create a Content-Length header to be able to keep a persistent HTTP connection. Note that HTTP rules prohibit a Content-Length header to be used in transfer-encoded messages.

- If AM-EL parameter or equivalent entity length information is not available, and HTTP rules allow for chunked transfer encoding, the OPES processor SHOULD use chunked transfer encoding. Note that any Content-Length header has to be removed in this case.

- If the message size is not known a priori and chunked transfer coding cannot be used, but the OPES processor can wait for the OCP transaction to finish before forwarding the adapted HTTP message on a persistent HTTP connection, then the processor SHOULD compute and add a Content-Length header.

- Finally, if all optimizations are not applicable, the OPES processor SHOULD delete any Content-Length header and forward adapted data immediately, while indicating the message end by closing the HTTP connection.

3.8.2. Content-MD5 Header

By default, the OPES processor MUST assume that the callout service modifies the content in a way that the MD5 checksum of the message body becomes invalid.

According to section 14.15 of [RFC2616], HTTP intermediaries must not generate Content-MD5 headers. A recalculation is therefore possible only if the OPES processor is considered authoritative for the entity being adapted. An un-authoritative OPES processor MUST remove the Content-MD5 header unless it detects that the HTTP message was not modified; in this case, it MAY leave the Content-MD5 header in the message. When such detection significantly increases message latency, deleting the Content-MD5 header may be a better option.

3.9. Examples

This is a possible OCP message flow using an HTTP request profile. An end-user wants to access the home page of www.restricted.example.com, through the proxy, but access is denied by a URL blocking service running on the callout server used by the proxy.

OCP messages from the OPES processor are marked with "P:" and OCP messages from the callout server are marked with "S:". The OCP connection is not closed at the end but kept open for the next OCP transaction.

Example:

```
 P: CS;
 S: CS;
 P: SGC 11 ({"31:ocp-test.example.com/url-filter"});
 P: NO ({"53:http://www.iana.org/assignments/opes/ocp/http/request"})
    SG: 11
    ;
 S: NR {"53:http://www.iana.org/assignments/opes/ocp/http/request"}
    SG: 11
    ;
 P: TS 55 11;
 P: AMS 55
    AM-EL: 0
    ;
 P: DUM 55 0
    Kept: 0
    AM-Part: request-header
    235:GET http://www.restricted.example.com/ HTTP/1.1
    Accept: */*
    Accept-Language: de
    Accept-Encoding: gzip, deflate
    User-Agent: Mozilla/4.0 (compatible; Windows NT 5.0)
    Host: www.restricted.example.com
    Proxy-Connection: Keep-Alive


    ;
 P: AME 55;
 S: AMS 55;
 S: DUM 55 0
    AM-Part: response-header
```

```
      76:HTTP/1.1 403 Forbidden
      Content-Type: text/html
      Proxy-Connection: close

      ;
   S: DUM 55 0
      AM-Part: response-body

      67:<html><body>You are not allowed to
      access this page.</body></html>
      ;
   S: AME 55;
   P: TE 55;
   S: TE 55;
```

Figure 13

The next example is a language translation of a small plain text file that gets transferred in an HTTP response. In this example, OCP agents negotiate a profile for the whole OCP connection. The OCP connection remains open in the end of the OCP transaction. (Note that NO and NR messages were rendered with an extra new line to satisfy RFC formatting requirements.)

Example:

```
   P: CS;
   S: CS;
   P: NO
      ({"54:http://www.iana.org/assignments/opes/ocp/http/response"});
   S: NR
      {"54:http://www.iana.org/assignments/opes/ocp/http/response"};
   P: SGC 12 ({"44:ocp-test.example.com/translate?from=EN&to=DE"});
   P: TS 89 12;
   P: AMS 89
      AM-EL: 86
      ;
   P: DUM 89 0
      AM-Part: response-header

      65:HTTP/1.1 200 OK
      Content-Type: text/plain
      Content-Length: 86


      ;
   P: DUM 89 65
      AM-Part: response-body
```

```
      86:Whether 'tis nobler in the mind to suffer
      The slings and arrows of outrageous fortune
      ;
   P: AME 89;
   S: AMS 89
      AM-EL: 78
      ;
   P: TE 89;
   S: DUM 89 0
      AM-Part: response-header

      65:HTTP/1.1 200 OK
      Content-Type: text/plain
      Content-Length: 78

      ;
   S: DUM 89 63
      AM-Part: response-body

      80:Ob's edler im Gemuet, die Pfeil und Schleudern
      des wuetenden Geschicks erdulden
      ;
   S: AME 89;
   S: TE 89;
```

Figure 14

The following example shows modification of an HTML resource and demonstrates data preservation optimization. The callout server uses a DUY message to send back an unchanged response header part, but because it does not know the size of the altered HTML resource at the time it sends the AMS message, the callout server omits the AM-EL parameter; the OPES processor is responsible for adjusting the Content-Length header.

Example:

```
   P: CS;
   S: CS;
   P: SGC 10 ({"30:ocp-test.example.com/ad-filter"});
   P: NO ({"54:http://www.iana.org/assignments/opes/ocp/http/response"
      Aux-Parts: (request-header,request-body)
      },{"45:http://www.iana.org/assignments/opes/ocp/MIME"})
      SG: 10
      ;
   S: NR {"54:http://www.iana.org/assignments/opes/ocp/http/response"
      Aux-Parts: (request-header)
      Content-Encodings: (gzip)
      }
```

```
      SG: 10
      ;
   P: TS 88 10;
   P: AMS 88
      AM-EL: 95
      ;
   P: DUM 88 0
      AM-Part: request-header

      65:GET /opes/adsample.html HTTP/1.1
      Host: www.martin-stecher.de


      ;
   P: DUM 88 65

      Kept: 65 64
      AM-Part: response-header

      64:HTTP/1.1 200 OK
      Content-Type: text/html
      Content-Length: 95


      ;
   P: DUM 88 129
      Kept: 65 90
      AM-Part: response-body

      26:<html>
      <body>
      This is my
      ;
   S: AMS 88;
   P: DUM 88 155
      Kept: 65 158
      AM-Part: response-body

      68: new ad: <img src="my_ad.gif"
      width=88 height=31>
      </body>
      </html>
      ;
   S: DUY 88 65 64
   S: DPI 88 129 2147483647;
   P: AME 88;
   S: DUM 88 0
      AM-Part: response-body
```

```
    52:<html>
    <body>
    This is my new ad:
    </body>
    </html>
    ;
 S: DPI 88 129 0;
 P: TE 88;
 S: AME 88;
 S: TE 88;
```

Figure 15

4. Tracing

[RFC3897] defines application-agnostic tracing facilities in OPES. Compliance with this specification requires compliance with [RFC3897]. When adapting HTTP, trace entries are supplied using HTTP message headers. The following HTTP extension headers are defined to carry trace entries. Their definitions are given using BNF notation and elements defined in [RFC2616].

```
OPES-System = "OPES-System" ":" #trace-entry
OPES-Via    = "OPES-Via" ":" #trace-entry

trace-entry = opes-agent-id *( ";" parameter )
opes-agent-id = absoluteURI
```

Figure 16

An OPES System MUST add its trace entry to the OPES-System header. Other OPES agents MUST use the OPES-Via header if they add their tracing entries. All OPES agents MUST append their entries. Informally, OPES-System is the only required OPES tracing header while OPES-Via provides optional tracing details; both headers reflect the order of trace entry additions.

If an OPES-Via header is used in the original application message, an OPES System MUST append its entry to the OPES-Via header. Otherwise, an OPES System MAY append its entry to the OPES-Via header. If an OPES System is using both headers, it MUST add identical trace entries except it MAY omit some or all trace-entry parameters from the OPES-Via header. Informally, the OPES System entries in the OPES-Via header are used to delimit and group OPES-Via entries from different OPES Systems without having a priory knowledge about OPES System identifiers.

Note that all of these headers are defined using #list constructs and, hence, a valid HTTP message may contain multiple trace entries per header. OPES agents SHOULD use a single header-field rather than using multiple equally-named fields to record a long trace. Using multiple equally-named extension header-fields is illegal from HTTP's point of view and may not work with some of the OPES-unaware HTTP proxies.

For example, here is an HTTP response message header after OPES adaptations have been applied by a single OPES processor executing 10 OPES services:

Example:

```
 HTTP/1.1 200 OK
 Date: Thu, 18 Sep 2003 06:25:24 GMT
 Last-Modified: Wed, 17 Sep 2003 18:24:25 GMT
 Content-type: application/octet-stream
 OPES-System: http://www.cdn.example.com/opes?session=ac79a749f56
 OPES-Via: http://www.cdn.example.com/opes?session=ac79a749f56,
     http://www.srvcs-4u.example.com/cat/?sid=123,
     http://www.srvcs-4u.example.com/cat/?sid=124,
     http://www.srvcs-4u.example.com/cat/?sid=125 ; mode=A
```

Figure 17

In the above example, the OPES processor has not included its trace entry or its trace entry was replaced by an OPES system trace entry. Only 3 out of 10 services are traced. The remaining services did not include their entries or their entries were removed by OPES system or processor. The last traced service included a "mode" parameter. Various identifiers in trace entries will probably have no meaning to the recipient of the message, but may be decoded by OPES System software.

OPES entities MAY place optional tracing entries in a message trailer (i.e., entity-headers at the end of a Chunked-Body of a chunked-encoded message), provided trailer presence does not violate HTTP protocol. See [RFC3897] for a definition of what tracing entries are optional. OPES entities MUST NOT place required tracing entries in a message trailer.

5. Bypass

An HTTP extension header is introduced to allow for OPES system bypass as defined in [RFC3897].

```
OPES-Bypass  = "OPES-Bypass" ":" ( "*" | 1#bypass-entry )
bypass-entry = opes-agent-id
```

Figure 18

This header can be added to HTTP requests to request OPES system bypass for the listed OPES agents. The asterisk "*" character is used to represent all possible OPES agents.

See [RFC3897] for what can be bypassed and for bypass requirements.

6. IAB Considerations

OPES treatment of IETF Internet Architecture Board (IAB) considerations [RFC3238] are documented in "OPES Treatment of IAB Considerations" [RFC3914].

7. Security Considerations

Application-independent security considerations are documented in application-agnostic OPES specifications. HTTP profiles do not introduce any HTTP-specific security considerations. However, that does not imply that HTTP adaptations are immune from security threats.

Specific threat examples include such adaptations as rewriting the Request-URI of an HTTP CONNECT request or removing an HTTP hop-by-hop Upgrade header before the HTTP proxy can act on it. As with any adaptation, the OPES agents MUST NOT perform such actions without HTTP client or server consent.

8. IANA Considerations

The IANA registers request and response profile features (Section 3.2) using the registration procedure outlined in the "IANA Considerations" Section of OCP Core [RFC4037]. The corresponding "uri" parameters for the two features are:

- http://www.iana.org/assignments/opes/ocp/http/request
- http://www.iana.org/assignments/opes/ocp/http/response

9. Compliance

Compliance with OPES mechanisms is defined in corresponding application-agnostic specifications. HTTP profiles for these mechanisms use corresponding compliance definitions from these specifications, as if each profile were incorporated into the application-agnostic specification it profiles.

10. References

10.1. Normative References

[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.

[RFC3897] Barbir, A., "Open Pluggable Edge Services (OPES) Entities and End Points Communication", RFC 3897, September 2004.

[RFC4037] Rousskov, A., "Open Pluggable Edge Services (OPES) Callout Protocol (OCP) Core", RFC 4037, March 2005.

10.2. Informative References

[RFC3835] Barbir, A., Penno, R., Chen, R., Hofmann, M., and H. Orman, "An Architecture for Open Pluggable Edge Services (OPES)", RFC 3835, August 2004.

[RFC3836] Beck, A., Hofmann, M., Orman, H., Penno, R., and A. Terzis, "Requirements for Open Pluggable Edge Services (OPES) Callout Protocols", RFC 3836, August 2004.

[RFC3837] Barbir, A., Batuner, O., Srinivas, B., Hofmann, M., and H. Orman, "Security Threats and Risks for Open Pluggable Edge Services (OPES)", RFC 3837, August 2004.

[RFC3752] Barbir, A., Burger, E., Chen, R., McHenry, S., Orman, H., and R. Penno, "Open Pluggable Edge Services (OPES) Use Cases and Deployment Scenarios", RFC 3752, April 2004.

[RFC3838] Barbir, A., Batuner, O., Beck, A., Chan, T., and H. Orman, "Policy, Authorization, and Enforcement Requirements of the Open Pluggable Edge Services (OPES)", RFC 3838, August 2004.

[rules-p] Beck, A. and A. Rousskov, "P: Message Processing Language", work in progress, October 2003.

[RFC3914] Barbir, A. and A. Rousskov, "Open Pluggable Edge Services (OPES) Treatment of IAB Considerations", RFC 3914, October 2004.

[RFC3238] Floyd, S. and L. Daigle, "IAB Architectural and Policy Considerations for Open Pluggable Edge Services", RFC 3238, January 2002.

Acknowledgements

The authors gratefully acknowledge the contributions of Robert Collins (Syncretize) and Larry Masinter (Adobe). Larry Masinter provided an early review of this document.

Authors' Addresses

Alex Rousskov
The Measurement Factory

EMail: rousskov@measurement-factory.com
URI: http://www.measurement-factory.com/

Martin Stecher
CyberGuard Corporation
Vattmannstr. 3
Paderborn 33100
DE

EMail: martin.stecher@webwasher.com
URI: http://www.webwasher.com/

Full Copyright Statement

Intellectual Property

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.