Network Working Group
Request for Comments: 2653
Category: Standards Track

J. Allen
WebTV Networks, Inc.
P. Leach
Microsoft
R. Hedberg
Catalogix
August 1999

CIP Transport Protocols

Status of this Memo

Copyright Notice

Abstract

This document specifies three protocols for transporting CIP requests, responses and index objects, utilizing TCP, mail, and HTTP. The objects themselves are defined in [CIP-MIME] and the overall CIP architecture is defined in [CIP-ARCH].

1. Protocol

In this section, the actual protocol for transmitting CIP index objects and maintaining the mesh is presented. While companion documents ([CIP-ARCH] and [CIP-MIME]) describe the concepts involved and the formats of the CIP MIME objects, this document is the authoritative definition of the message formats and transfer mechanisms of CIP used over TCP, HTTP and mail.

1.1 Philosophy

The philosophy of the CIP protocol design is one of building-block design. Instead of relying on bulky protocol definition tools, or ad-hoc text encodings, CIP draws on existing, well understood Internet technologies like MIME, RFC-822, Whois++, FTP, and SMTP. Hopefully this will serve to ease implementation and consensus

building. It should also stand as an example of a simple way to leverage existing Internet technologies to easily implement new application-level services.

1.2 Conventions

The key words "MUST" and "MAY" in this document are to be interpreted as described in "Key words for use in RFCs to Indicate Requirement Levels" [KEYWORDS].

Formal syntax is defined using ABNF [ABNF].

In examples octets sent by the sender-CIP are preceded by ">>> " and those sent by the receiver-CIP by "<<< ".

2 MIME message exchange mechanisms

CIP relies on interchange of standard MIME messages for all requests and replies. These messages are passed over a bidirectional, reliable transport system. This document defines transport over reliable network streams (via TCP), via HTTP, and via the Internet mail infrastructure.

The CIP server which initiates the connection (conventionally referred to as a client) will be referred to below as the sender-CIP. The CIP server which accepts a sender-CIP's incoming connection and responds to the sender-CIP's requests is called a receiver-CIP.

2.1 The Stream Transport

CIP messages are transmitted over bi-directional TCP connections via a simple text protocol. The transaction can take place over any TCP port, as specified by the mesh configuration. There is no "well known port" for CIP transactions. All configuration information in the system must include both a hostname and a port.

All sender-CIP actions (including requests, connection initiation, and connection finalization) are acknowledged by the receiver-CIP with a response code. See section 2.1.1 for the format of these codes, a list of the responses a CIP server may generate, and the expected sender-CIP action for each.

In order to maintain backwards compatibility with existing Whois++ servers, CIPv3 sender-CIPs MUST first verify that the newer protocol is supported. They do this by sending the following illegal Whois++ system command: "# CIP-Version: 3<cr><lf>". On existing Whois++ servers implementing version 1 and 2 of CIP, this results in a 500-series response code, and the server terminates the connection. If

the server implements CIPv3, it MUST instead respond with response code 300. Future versions of CIP can be correctly negotiated using this technique with a different string (i.e. "CIP-Version: 4"). An example of this short interchange is given below.

Note: If a sender-CIP can safely assume that the server implements CIPv3, it may choose to send the "# CIP-Version: 3" string and immediately follow it with the CIPv3 request. This optimization, useful only in known homogeneous CIPv3 meshes, avoids waiting for the roundtrip inherent in the negotiation.

Once a sender-CIP has successfully verified that the server supports CIPv3 requests, it can send the request, formatted as a MIME message with Mime-Version and Content-Type headers (only), using the network standard line ending: "<cr><lf>".

```
Cip-Req         = Req-Hdrs CRLF Req-Body

Req-Hdrs        = *( Version-Hdr | Req-Cntnt-Hdr )
Req-Body        = Body     ; format of request body as in [CIP-MIME]

Body            = Data CRLF "." CRLF
Data            =          ; data with CRLF "." CRLF replaced by
                           ; CRLF ".." CRLF

Version-Hdr     = "Mime-Version:" "1.0" CRLF
Req-Cntnt-Hdr   = "Content-Type:" Req-Content CRLF
Req-Content     =          ; format is specified in [CIP-MIME]

Cip-Rsp         = Rsp-Code CRLF [ Rsp-Hdrs CRLF Rsp-Body ]
                   [ Indx-Cntnt-Hdr CRLF Index-Body ]
Rsp-Code        = DIGIT DIGIT DIGIT Comment
Comment         =          ; any chars except CR and LF
Rsp-Hdrs        = *( Version-Hdr | Rsp-Cntnt-Hdr )
Rsp-Cntnt-Hdr   = "Content-Type:" Rsp-Content CRLF
Rsp-Content     =          ; format is specified in [CIP-MIME]
Rsp-Body        = Body     ; format of response body as in [CIP-MIME]

Indx-Cntnt-Hdr = "Content-Type:" Indx-Obj-Type CRLF
Indx-Obj-Type   =          ; any registered index object's MIME-type
                           ; the format is specified in [RFC2045]
Index-Body      = Body     ; format defined in each index
                           ; specifications

CRLF            =  CR LF   ; Internet standard newline
CR              =  %x0D    ; carriage return
LF              =  %x0A    ; linefeed
DIGIT           =  %x30-39
```

The message is terminated using SMTP-style message termination. The data is sent octet-for-octet, except when the pattern <cr><lf>1*["."]<cr><lf> is seen, in which case one more period is added.

When the data is finished, the octet pattern "<cr><lf>.<cr><lf>" is transmitted to the receiver-CIP.

On the receiver-CIP's side, the reverse transformation is applied, and the message read consists of all bytes up to, but not including, the terminating pattern.

In response to the request, the receiver-CIP sends a response code, from either the 200, 400, or 500 series. The receiver-CIP then processes the request and replies, if necessary, with a MIME message. This reply is also delimited by an SMTP-style message terminator.

After responding with a response code, the receiver-CIP MUST prepare to read another request message, resetting state to the point when the sender-CIP has just verified the CIP version. If the sender-CIP is finished making requests, it may close the connection. In response the receiver-CIP MUST abort reading the message and prepare for a new sender-CIP connection (resetting its state completely).

An example is given below. It is again worth reiterating that the command format is defined in [CIP-MIME] whereas the message body is defined in each index object definition. In this example the index object definition in [CIP-TIO] will be used. Line endings are explicitly shown in anglebrackets; newlines in this text are added only for readability. Comments occur in curly-brackets.

```
   { sender-CIP connects to receiver-CIP }
<<< % 220 Example CIP server ready<cr><lf>
>>> # CIP-Version: 3<cr><lf>
<<< % 300 CIPv3 OK!<cr><lf>
>>> Mime-Version: 1.0<cr><lf>
>>> Content-type: application/index.cmd.datachanged; type=
>>> x-tagged-index-1; dsi=1.2.752.17.5.10<cr><lf>
>>> <cr><lf>
>>> updatetype: incremental tagbased<cr><lf>
>>> thisupdate: 855938804<cr><lf>
>>> lastupdate: 855940000<cr><lf>
>>> .<cr><lf>
<<< % 200 Good MIME message received
>>> MIME-Version: 1.0<cr><lf>
>>> Content-Type: application/index.obj.tagged;
>>> dsi=1.2.752.17.5.10;
>>> base-uri="ldap://ldap.umu.se/dc=umu,dc=se"<cr><lf>
```

```
   >>> <cr><lf>
   >>> version: x-tagged-index-1<cr><lf>
   >>> updatetype: incremental<cr><lf>
   >>> lastupdate: 855940000<cr><lf>
   >>> thisupdate: 855938804<cr><lf>
   >>> BEGIN IO-schema<cr><lf>
   >>> cn: TOKEN<cr><lf>
   >>> sn: FULL<cr><lf>
   >>> title: FULL<cr><lf>
   >>> END IO-Schema<cr><lf>
   >>> BEGIN Update Block<cr><lf>
   >>> BEGIN Old<cr><lf>
   >>> title: 3/testpilot<cr><lf>
   >>> END Old<cr><lf>
   >>> BEGIN New<cr><lf>
   >>> title: 3/chiefpilot<cr><lf>
   >>> END New<cr><lf>
   >>> END Update Block<cr><lf>
   >>> .<cr><lf>
   <<< % 200 Good MIME message received
      { Sender-CIP shuts down socket for writing }
   <<< % 222 Connection closing in response to sender-CIP shutdown
      { receiver-CIP closes its side, resets, and awaits a
        new sender-CIP }
```

An example of an unsuccessful version negotiation looks like this:

```
      { sender-CIP connects to receiver-CIP }
   <<< % 220 Whois++ server ready<cr><lf>
   >>> # CIP-Version: 3<cr><lf>
   <<< % 500 Syntax error<cr><lf>
      { server closes connection }
```

The sender-CIP may attempt to retry using version 1 or 2 protocol. Sender-CIP may cache results of this unsuccessful negotiation to avoid later attempts.

2.1.1 Transport specific response codes

The following response codes are used with the stream transport:

Code	Suggested description text	Sender-CIP action
200	MIME request received and processed	Expect no output, continue session (or close)

201	MIME request received and processed, output follows	Read a response, delimited by SMTP-style message delimiter.
220	Initial server banner message	Continue with Whois++ interaction, or attempt CIP version negotiation.
222	Connection closing (in response to sender-CIP close)	Done with transaction.
300	Requested CIP version accepted	Continue with CIP transaction, in the specified version.
400	Temporarily unable to process request	Retry at a later time. May be used to indicate that the server does not currently have the resources available to accept an index.
500	Bad MIME message format	Retry with correctly formatted MIME
501	Unknown or missing request in application/index.cmd	Retry with correct CIP command
502	Request is missing required CIP attributes	Retry with correct CIP attributes.
520	Aborting connection for some unexpected reason	Alert local administrator.
530	Request requires valid signature	Sign the request, if possible, and retry. Otherwise, report problem to the administrator.
531	Request has invalid signature	Report problem to the administrator.
532	Cannot check signature	Alert local administrator, who should cooperate with remote administrator tp diagnose and resolve the problem. (Probably missing a public key.)

2.2 Internet mail infrastructure as transport

As an alternative to TCP streams, CIP transactions can take place over the existing Internet mail infrastructure. There are two motivations for this feature of CIP. First, it lowers the barriers to entry for leaf servers. When the need for a full TCP implementation is relaxed, leaf nodes (which, by definition, only send index objects) can consist of as little as a database and an indexing program (possibly written in a very high level language) to participate in the mesh.

Second, it keeps with the philosophy of making use of existing Internet technology. The MIME messages used for requests and responses are, by definition of the MIME specification, suitable for transport via the Internet mail infrastructure. With a few simple rules, we open up an entirely different way to interact with CIP servers which choose to implement this transport. See Protocol Conformance, below, for details on what options server implementers have about supporting the various transports.

The basic rhythm of request/response is maintained when using the mail transport. The following sections clarify some special cases which need to be considered for mail transport of CIP objects. In general, all mail protocols and mail format specifications (especially MIME Security Multiparts) can be used with the CIP mail transport.

2.2.1 CIP-Version negotiation

Since no information on which CIP-version is in use is present in the MIME message, this information has to be carried in the mailheader. Therefore CIP requests sent using the mail transport MUST include a CIP-version headerline, to be registered according to [MHREG]. The format of this line is:

```
DIGIT       =  %x30-39
number      =  1*DIGIT
cipversion  =  "CIP-Version:" <sp> number["." number]
```

2.2.2 Return path

When CIP transactions take place over a bidirectional stream, the return path for errors and results is implicit. Using mail as a transport introduces difficulties to the recipient, because it's not always clear from the headers exactly where the reply should go, though in practice there are some heuristics used by MUA's.

CIP solves this problem by fiat. CIP requests sent using the mail transport MUST include a Reply-To header as specified by RFC-822. Any mail received for processing by a CIP server implementing the mail transport without a Reply-To header MUST be ignored, and a message should be logged for the local administrator. The receiver MUST not attempt to reply with an error to any address derived from the incoming mail.

If there are no circumstances under which a response is to be sent to a CIP request, the sender should include a Reply-To header with the address "<>" in it. Receivers MUST never attempt to send replies to that address, as it is defined to be invalid (both here, and by the BNF grammar in RFC-822). It should be noted that, in general, it is a bad idea to turn off error reporting in this way. However, in the simplest case of an index pushing program, this MAY be a desirable simplification.

2.3 HTTP transport

HTTP MAY also be used to transport CIP objects, since they are just MIME objects. A transaction is performed by using the POST method to send an application/index.cmd and returning an application/index.response or an application/index.obj in the HTTP reply. The URL that is the target of the post is a configuration parameter of the CIP-sender to CIP-receiver relationship.

Example:

```
   { the client opens the connection and sends a POST }
>>> POST / HTTP/1.1<cr><lf>
>>> Host: cip.some.corp<cr><lf>
>>> Content-type: application/index.cmd.noop<cr><lf>
>>> Date: Thu, 6 Jun 1997 18:16:03 GMT<cr><lf>
>>> Content-Length: 2<cr><lf>
>>> Connection: close<cr><lf>
>>> <cr><lf>
   { the server processes the request }
<<< HTTP/1.1 204 No Content<cr><lf>
   { the server closes the connection }
```

In addition to leveraging the security capabilities that come with HTTP, there are other HTTP features that MAY be useful in a CIP context. A CIP client MAY use the Accept-Charset and Accept-Language HTTP headers to express a desire to retrieve an index in a particular character set or natural language. It MAY use the Accept-Encoding header to (e.g.) indicate that it can handle compressed responses, which the CIP server MAY send in conjunction with the Transfer-Encoding header. It MAY use the If-Modified-Since header to prevent

wasted transmission of an index that has not changed since the last poll. A CIP server can use the Retry-After header to request that the client retry later when the server is less busy.

3. Security Considerations

There are two levels at which the index information can be protected; the first is by use of the technology available for securing MIME [MIME-SEC] objects, and secondly by using the technology available for securing the transport.

When it comes to transport the stream transport can be protected by the use of TLS [TLS] . For HTTP the Security is handled by using HTTP Basic Authentication [RFC 2616], HTTP Message Digest Authentication [RFC2617] or SSL/TLS. Extra protection for the SMTP exchange can be achieve by the use of Secure SMTP over TLS [SMTPTLS].

4. References

[RFC 2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

[RFC 2616] 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.

[RFC 2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A. and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999.

[CIP-ARCH] Allen, J. and M. Mealling, "The Architecture of the Common Indexing Protocol (CIP)", RFC 2651, August 1999.

[CIP-MIME] Allen, J. and M. Mealling, "MIME Object Definitions for the Common Indexing Protocol (CIP)", RFC 2652, August 1999.

[ABNF] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

[CIP-TIO] Hedberg, R., Greenblatt, B., Moats, R. and M. Wahl, "A Tagged Index Object for use in the Common Indexing Protocol", RFC 2654, August 1999.

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

[MIME-SEC] Galvin, J., Murphy, S., Crocker, S. and N. Freed, "Security Multiparts for MIME: Multipart/Signed and Multipart/Encrypted", RFC 1847, October 1995.

[TLS] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[SMTPTLS] Hoffman, P., "SMTP Service Extension for Secure SMTP over TLS", RFC 2487, January 1999.

[MHREG] Jacob, P., "Mail and Netnews Header Registration Procedure", Work in Progress.

5. Authors' Addresses

Jeff R. Allen
246 Hawthorne St.
Palo Alto, CA 94301

EMail: jeff.allen@acm.org

Paul J. Leach
Microsoft
1 Microsoft Way
Redmond, WA 98052

EMail: paulle@microsoft.com

Roland Hedberg
Catalogix
Dalsveien 53
0775 Oslo
Norway

EMail: roland@catalogix.ac.se

6. Full Copyright Statement

Acknowledgement

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