Lemonade S. H. Maes
Internet Draft: Lemonade Bindings for firewalls R. Cromwell
and mobile network intermediaries N. Mitra
Informational Track (Editors)

Document: draft-ietf-lemonade-firewall-binding-00
Expires: August 2006 February 2006

Lemonade bindings to cross firewalls and mobile network intermediaries

Status of this Memo

Copyright Notice

Abstract

As part of the LEMONADE work to define extensions to the IMAP and SMTP protocols that provide optimizations in a variety of settings, the this document describes an alternative, optional binding for IMAP and SMTP showing how HTTP can be used to transfer commands and responses. This binding is intended to facilitate the use of IMAP and SMTP in deployments involving a variety of intermediaries. Bindings to SOAP, REST and WebDAV are also provided.

Conventions used in this document

In examples, "C:" and "S:" indicate lines sent by the client and server respectively.

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

An implementation is not compliant if it fails to satisfy one or more of the MUST or REQUIRED level requirements for the protocol(s) it implements. An implementation that satisfies all the MUST or REQUIRED level and all the SHOULD level requirements for a protocol is said to be "unconditionally compliant" to that protocol; one that satisfies all the MUST level requirements but not all the SHOULD level requirements is said to be "conditionally compliant." When describing the general syntax, some definitions are omitted as they are defined in [RFC3501], [RFC821], and related documents..

Table of Contents

1. Introduction and motivation

As part of the LEMONADE goal to define extensions to the IMAP and SMTP protocols [RFC3501] for providing optimizations in a variety of settings, this document describes how HTTP can optionally be used to transfer IMAP and SMTP commands and responses. This binding is intended to facilitate the use of IMAP and SMTP in deployments involving a variety of intermediaries, and offers a standardized alternative to de facto proprietary implementations of such a feature.

The need for an optional HTTP binding is driven by the needs of the mobile network operator community (see [MEMAIL][OMA-ME-RD]), where the reuse of an existing and well-understood technology will allow operators to apply their experience in solving practical deployment issues. Specifically, HTTP allows operators to reuse a similar setup and model that is already used for many other similar and related services, such as certain proprietary push e-mail and synchronization offerings, OMA Data Synchronization, Web services and Web access.

Using HTTP/HTTPS can simplify deployment in a corporate network through the potential use of a reverse proxy to achieve end-to-end encryption. This also has the advantage of not requiring changes to any firewall configurations and reduces the concerns that this often presents to corporation. In general the solution is compatible with any existing firewall. A reverse proxy can also support deployment models that offer roles to other service providers in the value chains, as discussed in [OMA-ME-AD].

The confidentiality, integrity, and compression capabilities used with HTTP and already implemented in a wide range of existing mobile device, can also be reused.

Studies have also shown that a persistent HTTP session has usually proven more resilient than an IMAP IDLE over TCP connection over an unreliable bearer such as a GPRS-based mobile network.

The use of HTTP as an underlying protocol for other application protocols has received much attention (see [RFC3205]). In particular, the concern exists that this circumvents firewall security policies. Another concern is the potential misuse or neglect of HTTP semantics by the application protocol that uses HTTP as a substrate.

Note that if the suppression of IMAP (or indeed any other application) traffic on HTTP/HTTPS is an issue, firewall administrators can still prevent such passage and this can provide incentives to re-configure firewalls to allow solutions on other transports (e.g. TLS) or offer the HTTP-based solution using another provisioned port (e.g. manually, out of band or via instructions like XGETLPREFS (see [NOTIFICATIONS])). The aim, therefore, is to allow for the use of this solution in the widest possible set of circumstances by codifying a standard way to do so that works with existing, deployed (i.e., HTTP only) firewalls, while explicitly allowing the possibility of detecting and filtering such traffic in deployments using the HTTP Content-Type in deployments where this is not permitted.

SOAP, REST and WeDAV binding are also described.

2. Techniques for binding over HTTP

There are two general approaches described below for binding IMAP over HTTP. The first approach shows how to tunnel regular IMAP requests and responses over HTTP using POST. The second approach proposes a syntactic change which recodes IMAP requests and responses as SOAP documents, WebDAV requests, or REST requests and attempts to obey the underlying semantics of those protocols. At the current stage of the draft, the SOAP, REST, and DAV mappings are meant more as informative examples for further research and discussion.

2.1. Tunneling Approaches

To use HTTP/HTTPS as the transfer protocol for IMAP commands and responses between the IMAP client and server, the client MUST send an HTTP POST request to the server, and embed IMAP commands (commands to an IMAPv4 Rev1 server or IMAP servers supporting Lemonade extensions) in the body of the request. A server MUST reject a HTTP GET request from the client. The content-type header of the POST request MUST be set to "application/vnd.lemonade". Multiple IMAP commands may be included in one POST request. In general, the HTTP server is expected to preserve session state between HTTP commands to the best of its ability, therefore the client does not need to reauthenticate and reissue a SELECT until it receives an (IMAP) error response showing that it is not authenticated.

In what follows, the term Lemonade client/server is used to refer to a client/server that supports both IMAPv4 Rev1 as well as any LEMONADE extensions.

When the HTTP binding is used, the Lemonade server listens on whatever port has been configured for this.

The following is an example of a possible Lemonade HTTP request:

```
   POST /lemonadePath HTTP/1.1 <CRLF>
   Content-Type: application/vnd.lemonade <CRLF>
   [other headers]
   <CRLF>
   (<tag> SP <Lemonade command> <CRLF> | literal )
   [(<tag> SP <Lemonade command> <CRLF> | literal )]
```

The Lemonade command MUST be plain text (7bit).

Multiple Lemonade commands MAY be sent on the same request. Thus Lemonade commands must be tagged. The client must be able to deal with recovering from errors when commands are batched. See RFC2442 Batch SMTP for a further discussion. In general, if a command is expected to produce a synchronized literal or continuation request, it MUST be the last command in the batch.

The Content-Type header is the only HTTP headers that MUST be sent to a Lemonade server. Other headers such as Cache-Control MAY be included.

When the Lemonade server sends back a response it is in following format:

```
   HTTP/1.1 <HTTP Status Code> <CRLF>
   Content-Type: text/plain <CRLF>
   <CRLF>
   [<untagged responses>]
   <tag> SP <Lemonade Server response> <CRLF>
   [<untagged responses>]
   <tag> SP <Lemonade Server response> <CRLF>
```

Notes:
The Lemonade Server uses the following HTTP status codes, and what each code indicates is given below:

- 100
  - This indicates the presence of a synchronizing literal or continuation request. The server is waiting for more data from the client (another HTTP request) before continuing. If the HTTP request includes batched commands after the command which generates a continuation request or synchronized literal, the server MUST generate a 5xx request.

- 200

- This indicates normal execution of the Lemonade commands from an IMAP perspective. The client should further parse the response body to get the tagged responses to the commands and process those accordingly.

- 401
  - This indicates that the execution of the IMAP commands might have been successful, but the session is no longer authenticated. The client should try to reauthenticate to the IMAP server, and then resend the commands.

- 5xx
  - This indicates that at least one command was malformed/protocol level error, or, a command could not complete due to a problem in the IMAP server. In conforming to HTTP semantics, this means the IMAP server responses such as BAD or NO on a tagged response generate a HTTP 500 response code.

When using HTTP to transfer IMAP commands and responses, the client SHOULD utilize built-in features of HTTP to their advantage. For example, the client SHOULD use HTTPS instead of HTTP whenever possible, since HTTPS has built in encryption and MAY have compression capabilities. STARTTLS should not be needed in this case, as it just requires additional overhead without any additional benefit.

HTTP can be used in both in-response and in-band modes. Details about these transport modes are given in the following two subsections.

2.1.1. Non-Persistent HTTP for In-response Connectivity Mode

If the client uses a traditional HTTP connection (either by establishing a different socket for each HTTP request to the Lemonade server, or by reusing the same socket for all HTTP requests, but sending each request under its own header), it has in-response connectivity to the server. The client can issue as many commands as it would like in one HTTP request to the server, and the server responds by sending back one HTTP response with all the responses to all the commands in the HTTP request. With this connectivity mode, the IDLE command cannot be issued. Other commands that use a continuation response or synchronized literal cannot be issued unless they are the last command in the batch. [LITERAL+] SHOULD be used to eliminate synchronized literals when using APPEND.

In order for the server to identify separate HTTP requests as belonging to the same session, an in-response HTTP client needs to

accept cookies. A session-id is passed in the cookie to identify the session.

Example: the headers for a HTTP In-response Response after the client has issued its first HTTP request to the server.

```
   HTTP/1.1 <HTTP Status Code> <CRLF>
   Content-Type: text/plain <CRLF>
   Set-Cookie:JSESSIONID=94571a8530d91e1913bfydafa;
path=/lemonade<CRLF>
   <CRLF>
   [<untagged responses>]
   <tag> SP <Lemnade Server response> <CRLF>
   [[<untagged responses>]
   <tag> SP <Lemonade Server response> <CRLF>]
```

Example: the headers for a HTTP In-response Response after the client has issued its first HTTP request to the server, with the final command generating a continuation request.

```
   HTTP/1.1 100 Continue <CRLF>
   Content-Type: text/plain <CRLF>
   Set-Cookie:JSESSIONID=94571a8530d91e1913bfydafa;
path=/lemonade<CRLF>
   <CRLF>
   [<untagged responses>]
   <tag> SP <Lemnade Server response> <CRLF>
   +continuation-request
```

The client must then save this cookie and send it back to the server with the next request in order for the server to reattach these commands to the same session as the previous commands.

```
   POST /lemonadePath HTTP/1.1 <CRLF>
   Content-Type: application/vnd.lemonade <CRLF>
   Cookie: JSESSIONID=94571a8530d91e1913bfydafa
   [other headers]
   <CRLF>
   <tag> SP <Lemonade command> <CRLF>
   [<tag> SP <Lemonade command> <CRLF>]
```

2.1.2. Using Persistent HTTP/HTTPS + Chunked Transfer Encoding for In-band Connectivity Mode

It is possible to use persistent HTTP or persistent HTTPS plus chunked- transfer-encoding so that the server can instantly send

notifications to the client while a session is open. The client needs to open a persistent connection and keep it active. In this case, the HTTP headers must be sent the first time the client device opens the connection to the Lemonade Server and these headers MUST set the transfer coding to be chunk-encoded [RFC2616, Sec. 3.6.1]. All subsequent client-server requests are written to the open connection, without needing any additional headers negotiations. The server can use this open channel to push events to the client device at any time. In this case, the client SHOULD NOT accept cookies.

The client must send the HTTP headers one time only:

```
   POST /lemonadeServletPath HTTP/1.1 <CRLF>
   Content-Type: application/vnd.lemonade <CRLF>
   Connection: keep-alive <CRLF>
   Pragma: no-cache <CRLF>
   Transfer-Encoding: chunked <CRLF>
```

The server responds with the following header:

```
   HTTP/1.1 <HTTP Status Code> <CRLF>
   Cache-Control: private
   Keep-Alive: timeout=15, max=100 (or other suitable setting)
   Connection: Keep-Alive
   Transfer-Encoding: chunked
   Content-Type: text/plain
```

Then the client can send a command anytime it wants with the following format:

```
   <length of Lemonade command, including bytes in CRLF> <CRLF>
   <tag> SP <Lemonade command> <CRLF>
   <CRLF>
```

And example of an actual client command is:

```
   e <CRLF>
   2 CAPABILITY<CRLF>
   <CRLF>
```

The server responds to each command with as many untagged responses as needed, and one tagged response, where each response is in the format that follows:

```
   <length of a single response, including bytes in CRLF> <CRLF>
   <tagged or untagged response> <CRLF>
   <CRLF>
```

An actual Server response might be:

```
   d5 <CRLF>
```

```
   * CAPABILITY IMAP4REV1 AUTH=LOGIN NAMESPACE SORT MULTIAPPEND
LITERAL+ UIDPLUS IDLE XORACLE X-ORACLE-LIST X-ORACLE-COMMENT X-
ORACLE-QUOTA X-ORACLE-PREF X-ORACLE-MOVE X-ORACLE-DELETE ACL X-
ORACLE-PASSWORD LDELIVER LZIP LCONVERT LFILTER LSETPREF LGETPREF
<CRLF>   <CRLF>
   1b <CRLF>
   2 OK CAPABILITY completed <CRLF>
   <CRLF>
```

Note however that the HTTP protocol is in general not meant to be used in such a way. To maintain such an open channel might be a practical challenge to proxies/firewalls, which might not forward the requests chunk by chunk to the server, and meanwhile route responses back to the client chunk by chunk. Consequently the session closes. Chunked transfer encoding requests MAY not be honored by an HTTP server. In cases where such requests are denied, the client should be prepared to use the non-chunked encoding technique from section 2.1

The same challenges exist for TCP session.

In any case, the session can be automatically started again by the client after a lost connection or by the server through out-of-band; after some defined time-out.

2.1.3. Using HTTP Connect

If a HTTP proxy server is available to the client which supports the HTTP CONNECT method, and the IMAP server the user wishes to reach allows external connections outside the destination network s firewall, the client may wish to tunnel a regular TCP connection through the HTTP proxy.

See [LUOTONEN] or section 5.2 of [RFC2817] for a detailed description of the technique. Note that HTTP Proxy servers may not honor all CONNECT requests, and may in fact, limit CONNECT requests to a small number of common ports, such as 80, 443, 8080, etc. It is advised that networks wishing to allow their users to use this feature allow clients within their network to CONNECT to ports 25, 143, 587, and 993.

2.1.4. Using HTTP as a binding for SMTP

All of the techniques described in sections 2.1, 2.2, and 2.3 may be used for SMTP as well. The only difference between IMAP and SMTP will be the HTTP URL used. Servers implementing the HTTP binding are

expected to differentiate between IMAP and SMTP protocol bodies via the URL.

2.2. Syntactic Mapping Approaches

The following mappings shows how synthactic mapping approaches can be used to map IMAP /SMTO over SOAP, REST, and WebDAV.

2.3. Using SOAP (Web Services) as a binding for IMAP

The SOAP binding attempts to map IMAP commands to SOAP methods, and IMAP data types and grammar (atoms, lists, et al) to document-literals supplied as the soap body. This is essentially a tunneling technique with a syntactic change. The following general encoding rules are proposed:

- IMAP commands are translated into SOAP methods of the same name, e.g. the FETCH command becomes the FETCH SOAP method name. (UID FETCH is mapped to UID_FETCH).
- SOAP document literal style is used
- Terminals in the IMAP grammar which represent atoms become elements (e.g. FLAGS becomes <FLAGS/>). Flags are stripped of leading backslash and uppercased.
- Non-terminals which are an ATOM followed by a single parameter are represented as a non-empty element containing that parameter(e.g. CHARSET foo becomes <CHARSET>foo</CHARSET>, or SENTBEFORE date becomes <SENTBEFORE>date</SENTBEFORE>).
- Lists are represented as <L> </L> containing zero or more elements (including other <L>s)
- Unless otherwise defined, if a particular keyword is followed by more than one value, each value is encoded as <P>value</P> as placed as a child element. E.g. APPEND mailbox SP flaglist SP literal becomes
<APPEND><P>mailbox</P><P><L><ANSWERED/><DRAFT/></L></P></APPEND>
- Continuation responses and requests are encapsulated as <C>data</C>
- Literals are encapsulated as <T>text</T> or <B>binary</B>
- Unsolicited responses are encapsulates as <U>response</U>
- The partial specifier is <P>offset.length</P>
- The section specifier is <SECTION> </SECTION>
- A sequence set is wrapped as <SEQUENCE>sequence-set</SEQUENCE>
- The IMAP response is encoded in <RESP>response</RESP>
- Any responses which start with a number followed by an ATOM are encoded as <ATOM>number</ATOM>

The following is an example encoding:

C: a1 FETCH 1:5,9 BODY[1.1.CONVERT(TEXT/PLAIN)]<1024.2048>

Becomes

```
<FETCH>
   <SEQUENCE>1:5,9</SEQUENCE>
  <BODY>
      <SECTION>
         <P>1.1.CONVERT( TEXT/PLAIN )</P>
       </SECTION>
      <P>1024.2048</P>
   </BODY>
</FETCH>
```

This would then be invoked on a Web Service via the SOAPMethodName FETCH . The expected response would be zero or more <U> elements containing <FETCH> elements which encode the returned data.

These rules are by no means complete and exhaustive, and more stringent encoding rules are needed to encompass the full range of IMAP extended ABNF. The above rules are provided as a starting point.

SOAP by itself adds considerable overhead to requests, so it would not be recommended without some form of compression or compact encoding such as Fast Web Services (X.695 ASN.1 Support for SOAP, Web Services and the XML Information Set)[X.695]. However, SOAP may provide some benefits over raw HTTP for those who have existing investments in SOAP infrastructure.

Usage of X.695 is optional.

As a final note, the above usage once again, assumes that the SOAP server is not stateless and uses HTTP cookies to preserve IMAP session state between requests.

Here s an example session side by side with IMAP syntax(SOAP envelop not shown):

```
C-SOAP: <LOGIN><P>username</P><P>password</P>
C-IMAP: a1 LOGIN username password

S-SOAP: <RESP><OK>LOGIN Ok</OK>
S-IMAP: * OK LOGIN Ok

C-SOAP: <SELECT>INBOX</SELECT>
C-IMAP: a2 SELECT INBOX

S-SOAP: <RESP>
   <U>
      <FLAGS><L>
               <ANSWERED/>
               <DRAFT/>
```

```
            <FLAGGED/>
            <SEEN/>
         </L>
      </FLAGS>
   </U>
   <U>
      <OK>
      <PERMANENTFLAGS>
         <L>
            <ANSWERED/>
            <DRAFT/>
            <FLAGGED/>
            <SEEN/>
         </L>
      </PERMANENTFLAGS>
      </OK>
   </U>
   <U>
      <EXISTS>1234</EXISTS>
   </U>
   <U>
      <RECENT>0</RECENT>
   </U>
   <U>
      <OK>
         <UIDVALIDITY>12345678</UIDVALIDITY>
      </OK>
   </U>
   <OK><READ-WRITE/></OK>
   </RESP>

S-IMAP: * FLAGS (\Answered \Draft \Flagged \Seen)
S-IMAP: * OK [PERMANENTFLAGS (\Answered \Draft \Flagged \Seen)]
S-IMAP: * 1234 EXISTS
S-IMAP: * 0 RECENT
S-IMAP: * Ok [UIDVALIDITY 12345678]
S-IMAP: a2 OK [READ-WRITE]
```

2.4. REST Mapping

[REST] stands for Representation State Transfer, and is an architectural style modeled on HTTP, which seeks to build applications around the elements of HTTP s design which are attributed to its wide success and large scalability.

The tunneling approach in section 2.1 violates REST principles because it doesn t model server state as resources and doesn t seek to use the underlying HTTP operations according to their true semantics.

REST suggests that server resources should be modeled as, and addressable as URLs, instead of as the result of the execution of verbs. SOAP RPC seeks to model manipulation of resources as the invocation of a method which returns the resource, such as executeFetch , whereas REST seeks to model those resources via a uniform interface (a URL), that can be manipulated via standard HTTP commands.

To create a mapping of IMAP to RESTful HTTP, a discussion entailing the description of what resources IMAP exports, what uniform interface will be used to locate those resources, and what representation will be used to exchange those resources (e.g.) must be provided.

2.4.1. IMAP resources as REST resources and interface

An IMAP server primary consists of mailboxes and messages. A mailbox contains a collection of messages, and a message contains message contents. Both mailboxes and messages also have server specified metadata attached, such as flags, annotations, etc

An Example REST interface to such data, might take the form of the following examples:

http://imap.server.com/mailboxname/

To refer to a mailbox resource, and

http://imap.server.com/mailboxname/messageuid

To refer to a message in a mailbox.

Metadata about a mailbox or message might be identified as

http://imap.server.com/mailboxname/annotations

or

http://imap.server.com/mailboxname/messageuid/flags

Message body parts might be represented via a hierarchical URL syntax, such as

http://imap.server.com/mailboxname/messageuid/body/1/2/3
(BODY[1.2.3])

or with convert (BODY[1.2.3.CONVERT (image/gif ..)]

http://imap.server.com/mailboxname/messageuid/body/1/2/3/convert/image/gif

2.4.2. IMAP commands as HTTP commands on REST resources

REST generally views GET requests as idempotent or requests that do not mutate a resource, PUT requests as storing a new resource at the specified URL, DELETE as removing resources located by the URL, and POST as potentially performing some server defined action on the specified resource.

Given the above guidelines, IMAP commands such as FETCH (with BODY.PEEK or BINARY.PEEK) would be considered as GET requests, commands such as STORE and APPEND would be considered candidates for PUT mapping, and commands such as EXPUNGE, CREATE, or RENAME might be modeled as POST.

Commands which may return multiple resources (UID FETCH n-m,x-y) may be modeled as a collection resource with a query, such as

http://imap.server.com/mailboxname/allmsgs?uids=n-m,x-y

An IMAP immediate delete of a single message can be carried out via REST via a HTTP DELETE of the URL identifying that message. However, an IMAP delete of several messages by marketing \Deleted, followed by an expunge, would have to be carried out via several PUT requests to set the flags on a particular message, followed by an EXPUNGE via POST.

Because REST frowns on the use of PUT with query parameters, a multi-update of several messages at once with the same flags, would either require multiple PUTs (one per message), or a new POST URL which takes a collection URL and performs the operation, such as

POST http://imap.server.com/mailboxname/storeflags?uids=n-m,x-y
(body of request indicating that \Deleted is the flag to be updated)

2.4.3. Representation of transferred resources

REST does not dictate the usage of XML. Because of this, a REST binding could in fact use IMAP responses for its syntax. A GET request of http://imap.server.com/mailboxname/ for example, could act as FETCH 1:* UID and return the untagged * FETCH responses from server.

A GET request on a message resource could simply return RFC822 format text, for example.

2.4.4. Challenges

The challenge of producing a REST binding for IMAP lies not in mapping IMAP resources to HTTP URLs, but of allowing the client to take advantage of efficient IMAP commands, such as fetching a subset of data over a subset of a collection of messages (SEARCH and FETCH commands) in a way that preserves the REST model as much as possible. Also, mapping IMAP security, and IMAP extensions at this point, remains a challenge and has to be done on a case by case basis.

Unlike the SOAP binding, which is a mere syntax transformation of IMAP, producing REST notions of arbitrary IMAP extensions is an unbounded scope of work. It may help however, to consider only the set of extensions that MUST be implemented in Lemonade Profile Phase 2 as the candidates for mapping, and work from there.

2.5. WebDAV Mapping

WebDAV models collections of resources with structured metadata in XML form via a URL abstraction, with typical operations such retrieval, copy, delete, move, and update. It is REST-like, with additional semantics related to metadata.

WebDAV differs from REST in that it adds a more rigorous definition of what request and response payloads are, specifically to manipulate metadata properties, as well as defining the concept of a collection of resources. WebDAV also adds new HTTP methods such as COPY and MOVE.

Existing WebDAV mappings for IMAP already exist. Microsoft Outlook contains such a mapping for HotMail, referred to as HTTPMail, which treats IMAP mailboxes as WebDAV collections.

The approach suggested here is similar, which is to model IMAP mailboxes as WebDAV collections, with mailbox specific metadata treated as WebDAV metadata properties about the resource (EXISTS, UIDNEXT, etc). Messages within a mailbox are treated as resources within a WebDAV collection. Message envelope and other metadata are modeled as WebDAV properties attached to the resource.

Many IMAP commands can be mapped to WebDAV commands which manipulate collections, however, due to differences in the underlying semantics of WebDAV and the lack of some operations that exist in Lemonade which do not in WebDAV, a sufficient mapping at this time is not possible.

For example, IMAP APPEND can be mapped to WebDAV PUT, and IMAP STORE can be mapped to WebDAV PROPPATCH, but Lemonade CATENATE cannot be mapped to any WebDAV sequence, because WebDAV lacks the ability to append to an existing resource (it can only overwrite it), and the WebDAV COPY command cannot take multiple source arguments. IMAP SEARCH can t be mapped unless one takes into account the draft WebDAV SEARCH command.

Moreover, WebDAV s security model with respect to authorization differs from IMAP further complicating a mapping, and IMAP extensions like CONVERT would have to be mapped outside the bounds of the DAV spec via HTTP POST.

As such, a strict WebDAV mapping would have to be a subset of Lemonade Profile. Therefore, a complete mapping must combine the approaches of REST using POST to map actions, and WebDAV for resources for which a good mapping already exists.

3. Security Considerations

HTTP binding has the same security requirements as IMAP when using an in-response or inband connectivity mode.

The HTTPS protocol can be used to provide end-to-end security

Proxy-based implementations may still require payload encryption for end-to-end security.

Caching is a concern. The client SHOULD use the HTTP Cache-Control directive (no-cache, no-store, must-revalidate, or combinations thereof) to inform proxy servers, origin servers, and client libraries not to cache or store the HTTP response. To deal with HTTP 1.0 servers that may exist in the network, Pragma: no-cache should be

used as well.

Attacks on HTTP sessions and the HTTP server may also be a concern, since the HTTP server is maintaining an authenticated session to the IMAP server on behalf of the user in most cases.

Firewall administrators wishing to block stealth deployments of HTTP IMAP bindings may block HTTP requests with Content-Type application/vnd.lemonade via an application level firewall.

4. References

[LEMONADEPROFILE] Maes, S.H. and Melnikov A., "Lemonade Profile", draft-ietf-lemonade-profile-XX.txt, (work in progress).

[LUOTONEN] Luotonen, A., Tunneling TCP based protocols through Web proxy servers , draft-luotonen-web-proxy-tunneling-01.txt, August 1998

[MEMAIL] Maes, S.H., Lemonade and Mobile e-mail", draft-maes-lemonade-mobile-email-xx.txt, (work in progress).

[NOTIFICATIONS] Maes, S.H., Lima R., Kuang, C., Cromwell, R., Ha, V. and Chiu, E., Day, J., Ahad R., Jeong W-H., Rosell G., Sini, J., Sohn S-M., Xiaohui F. and Lijun Z., "Server to Client Notifications and Filtering", draft-ietf-lemonade-server-to-client-notifications-xx.txt, (work in progress).

[OMA-ME-AD] Open Mobile Alliance Mobile Email Architecture Document, (Work in progress). http://www.openmobilealliance.org/

[OMA-ME-RD] Open Mobile Alliance Mobile Email Requirement Document, (Work in progress). http://www.openmobilealliance.org/

[P-IMAP] Maes, S.H., Lima R., Kuang, C., Cromwell, R., Ha, V. and Chiu, E., Day, J., Ahad R., Jeong W-H., Rosell G., Sini, J., Sohn S-M., Xiaohui F. and Lijun Z., "Push Extensions to the IMAP Protocol (P-IMAP)", draft-maes-lemonade-p-imap-xx.txt, (work in progress).

[REST] Fielding, Roy Thomas. Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine, 2000.

[RFC2088] Myers, J. IMAP non-synchronizing literals , RFC2088, January 1997
http://www.ietf.org/rfc/rfc2088

   <Lemonade binding for firewalls and mobile networks> February 2006

[RFC2119] Brader, S. "Keywords for use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997. http://www.ietf.org/rfc/rfc2119

[RFC2442] Freed, N. et al. "The Batch SMTP Media Type", RFC 2442, November 1998. http://www.ietf.org/rfc/rfc2442

[RFC2616] Fielding, R. et al. "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999. http://www.ietf.org/rfc/rfc2616

[RFC2817] Khare, R., Upgrading to TLS Within HTTP/1.1 , RFC2817, May 2000 http://www.ietf.org/rfc/rfc2817.txt, May 2000

[RFC3205] Moore, K. On the use of HTTP as a Substrate , RFC 3205, February 2002. http://www.ietf.org/rfc/rfc3205

[RFC3501] Crispin, M. "IMAP4, Internet Message Access Protocol Version 4 rev1", RFC 3501, March 2003. http://www.ietf.org/rfc/rfc3501

[X.695] X.695 ASN.1 Support for SOAP, Web Services and the XML Information Set , ITU/ISO http://java.sun.com/developer/technicalArticles/WebServices/fastWS/

[WEBDAV] Goland, Y., Whitehead, E., Faizi, A., Carter, S.R., and D. Jensen, HTTP Extensions for Distributed Authoring -- WEBDAV , RFC 2518, February 1999

.

5. Future Work

TBD[1] Should an OPTIONS HTTP request be supported to allow a client to probe HTTP binding capabilities, such as which protocol a given URL is bound to, or whether chunking is supported?

[2] Should separate content types exist for IMAP and SMTP since the entity body in the HTTP request is different?

[3] Standardizing the form of the URL for the binding may permit firewall administrations to impose better filtering.

[4] Produce more rigorous rules for mapping IMAP and SMTP ABNF to SOAP, REST, and DAV.

[5] Provide ways to declare supported bindings or select a binding.

6. Version History

Release 00

Initial release published in February 2006. Carried over from draft-maes-lemonade-http-binding-04 and now made into a working group document. Added REST and WebDAV binding discussion. Clarified HTTP response codes.

Acknowledgments

The authors want to thank all who have contributed key insight and extensively reviewed and discussed the concepts of HTTP Bindings and its early introduction in P-IMAP [P-IMAP].

Authors Addresses

Stephane H. Maes
Oracle Corporation
500 Oracle Parkway
M/S 4op634
Redwood Shores, CA 94065
USA
Phone: +1-650-607-6296
Email: stephane.maes@oracle.com

Ray Cromwell
Oracle Corporation
500 Oracle Parkway
Redwood Shores, CA 94065
USA

Nilo Mitra
Ericsson
Tel: +1 212-843-8451
Email: nilo.mitra@ericsson.com

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgement

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