INTERNET-DRAFT
Expires: August 1998
<draft-luotonen-web-proxy-tunneling-00.txt>
February 1998

Ari Luotonen
Netscape Communications Corporation

Tunneling TCP based protocols through Web proxy servers

Status of this Memo

Abstract

This document specifies a generic tunneling mechanism for TCP based protocols through Web proxy servers. This tunneling mechanism was initially introduced for the SSL protocol [SSL] to allow secure Web traffic to pass through firewalls, but its utility is not limited to SSL. Earlier drafts of this specification were titled ''Tunneling SSL through Web Proxy Servers'' <draft-luotonen-ssl-tunneling-XX.txt>. Implementations of this tunneling feature are commonly referred to as ''SSL tunneling'', although, again, it can be used for tunneling any TCP based protocol.

A wide variety of existing client and proxy server implementations conform to this specification. The purpose of this specification is to describe the current practice, to propose some good practices for implementing this specification, and to document the security considerations that are involved with this protocol.

Table of Contents

1. Overview

The wide success of the SSL (Secure Sockets Layer) protocol made it vital for Web proxy servers to be able to tunnel requests performed over SSL. The easiest, and perhaps the most elegant, way to accomplish this is to extend the HTTP/1.x protocol [HTTP/1.0, HTTP/1.1] in such a way that it will be able to intiate a tunnel through the proxy server.

This document specifies the HTTP/1.x extension to implement the generic TCP protocol tunneling on Web proxy servers. This extension may be used between clients and proxy servers, and between two proxies (in the case of daisy-chained proxies -- proxies that contact other proxies to perform requests). This document focuses on the differences and additions to HTTP/1.x; refer to the HTTP/1.x specifications for a full specification of HTTP/1.x.

Note that the HTTPS protocol, which is just HTTP on top of SSL, could alternatively be proxied in the same way that other protocols are handled by the proxies: to have the proxy (instead of the client) initiate the secure session with the remote HTTPS server, and then perform the HTTPS transaction on the client's part. The response will be received and decrypted by the proxy, and sent to the client over (insecure) HTTP. This is the way FTP and Gopher get handled by proxies. However, this approach has several disadvantages and complications:

* The connection between the client and the proxy is normal HTTP, and hence, not secure. This may, however, often be acceptable if the clients are in a trusted subnetwork (behind a firewall).

* The proxy will need to have full SSL implementation incorporated into it -- something this tunneling mechanism does not require.

* The client will not be able to perform SSL client authentication (authentication based on X509 certificates) to the remote server, as the proxy will be the authenticated party. Future versions of SSL may, however, provide such delegated authentication.

This specification defines a tunneling mechanism for Web proxy servers. This mechanism is compatible with HTTP/1.x protocol, which is currently being used by Web proxies.

Note that this mechanism, if used for SSL tunneling, does not require an implementation of SSL in the proxy. The SSL session is established between the client generating the request, and the destination (secure) Web server; the proxy server in between is merely tunneling the encrypted data, and does not take any other part in the secure transaction.

2. General Considerations with Respect to SSL Tunneling

When tunneling SSL, the proxy must not have access to the data being transferred in either direction, for the sake of security. The proxy merely knows the source and destination addresses, and possibly, if the proxy supports user authentication, the name of the requesting user.

In other words, there is a handshake between the client and the proxy to establish the connection between the client and the remote server through the proxy. In order to make this extension be backward compatible, the handshake must be in the same format as HTTP/1.x requests, so that proxies without support for this feature can still cleanly determine the request as impossible for them to service, and give proper error responses (rather than e.g. get hung on the connection).

3. Functional Specification

3.1. Request

The client connects to the proxy server, and uses the CONNECT method to specify the hostname and the port number to connect to. The hostname and port number are separated by a colon, and both of them must be specified.

The host:port part is followed by a space and a string specifying the HTTP version number, e.g. HTTP/1.0, and the line terminator (CR LF

pair. Note that some applications may use just a LF on its own, and it is recommended that applications be tolerant of this behavior. When this document refers to CR LF pair, in all cases should a LF on its own be treated the same as a CR LF pair).

After that there is a series of zero or more of HTTP request header lines, followed by an empty line. Each of those header lines is also terminated by the CR LF pair. The empty line is simply another CR LF pair.

After the empty line, if the handshake to establish the connection was successful, the tunnelled (SSL or other) data transfer can begin. Before the tunneling begins, the proxy will respond, as described in the next section (Section 3.2).

Example of an SSL tunneling request to host home.netscape.com, to HTTPS port (443):

```
CONNECT home.netscape.com:443 HTTP/1.0
User-agent: Mozilla/4.0

...data to be tunnelled to the server...
```

Note that the "...data to be tunnelled to the server..." is not a part of the request. It is shown here only to make the point that once the tunnel is established, the same connection is used for transferring the data that is to be tunnelled.

The advantage of extending the HTTP/1.x protocol in this manner (a new method) is that this protocol is freely extensible just like HTTP/1.x is. For example, the proxy authentication may be used just like with any other request to the proxy:

```
CONNECT home.netscape.com:443 HTTP/1.0
User-agent: Mozilla/4.0
Proxy-authorization: basic dGVzdDp0ZXN0

...data to be tunnelled to the server...
```

3.2. Proxy Response

After the empty line in the request, the client will wait for a response from the proxy. The proxy will evaluate the request, make sure that it is valid, and that the user is authorized to request such a connection. If everything is in order, the proxy will make a connection to the destination server, and, if successful, send a "200 Connection established" response to the client. Again, the response

follows the HTTP/1.x protocol, so the response line starts with the protocol version specifier, and the response line is followed by zero or more response headers, followed by an empty line. The line separator is CR LF pair.

Example of a response:

```
      HTTP/1.0 200 Connection established
      Proxy-agent: Netscape-Proxy/1.1

      ...data tunnelled from the server...
```

After the empty line, the proxy will start passing data from the client connection to the remote server connection, and vice versa. At any time, there may be data coming from either connection, and that data must be forwarded to the other connection immediately.

Note that since the tunnelled protocol is opaque to the proxy server, the proxy cannot make any assumptions about which connection the first, or any subsequent, packets will arrive. In other words, the proxy server must be prepared to accept packets from either of the connections at any time. Otherwise, a deadlock may occur.

If at any point either one of the peers gets disconnected, any outstanding data that came from that peer will be passed to the other one, and after that also the other connection will be terminated by the proxy. If there is outstanding data to that peer undelivered, that data will be discarded.

An example of a tunneling request/response in an interleaved multicolumn format:

```
     CLIENT -> SERVER                        SERVER -> CLIENT
     --------------------------------------
-----------------------------------
     CONNECT home.netscape.com:443 HTTP/1.0
     User-agent: Mozilla/4.0
     <<< empty line >>>
                                             HTTP/1.0 200 Connection
established
                                             Proxy-agent: Netscape-Proxy/1.1
                                             <<< empty line >>>
                  <<< data tunneling to both directions begins >>>
```

3.2.1. Response Content-Type Field

The proxy response does not necessarily have a Content-Type field, which is otherwise mandatory in HTTP/1.x responses. Currently there is no content media type assigned to a tunnel. Future versions of this specification may introduce a standard media type, for example "application/tunnel". For forward compatibility, a Content-type field should be allowed, but for backward compatibitity, one should not be required by clients.

3.3. Data Pipelining

It is legal for the client to send some data intended for the server before the "200 Connection established" (or any other success or error code) is received. This allows for reduced latency and increased efficiency when any handshake data intended for the remote server can be sent in the same TCP packet as the proxy request. This allows the proxy to immediately forward the data once the connection to the remote server is established, without waiting for two round-trip times to the client (sending 200 to client; waiting for the next packet from client).

This means that the proxy server cannot assume that reading from the client socket descriptor would only return the proxy request. Rather, there may be any amount of opaque data following the proxy request that must be forwarded to the server once the connection is established. However, if the connection to the remote server fails, or if it is disallowed by the proxy server, the data intended to the remote server will be discarded by the proxy.

At the same time this means that a client pipelining data intended for the remote server immediately after sending the proxy request (or in the same packet), must be prepared to re-issue the request and re-compose any data that it had already sent, in case the proxy fails the request, or challenges the client for authentication credentials. This is due to the fact that HTTP by its nature may require the request to be re-issued, accompanied by authentication credentials or other data that was either missing or invalid in the original request.

Note that it is not recommended to pipeline more data than the amount that can fit to the remainder of the TCP packet that the proxy request is in. Pipelining more data can cause a TCP reset if the proxy fails or challenges the request, and subsequently closes the connection before all pipelined TCP packets are received by the proxy server host. A TCP reset will cause the proxy server's response to be discarded, and not be available to the client -- thus being unable to determine whether the failure was due to a network error, access control, or an authentication challenge.

4. Extensibility

The tunneling handshake is freely extensible using the HTTP/1.x headers; as an example, to enforce authentication for the proxy the proxy will simply use the 407 status code and the Proxy-authenticate response header (as defined by the HTTP/1.x specification) to ask the client to send authentication information:

```
      HTTP/1.0 407 Proxy authentication required
      Proxy-authenticate: ...
```

The client would then reperform the request, and send the authentication information in the Proxy-authorization header:

```
      CONNECT home.netscape.com:443 HTTP/1.0
      User-agent: ...
      Proxy-authorization: ...

      ...data to be tunnelled to the server...
```

The full example displayed in an interleaved multicolumn format:

```
     CLIENT -> SERVER                        SERVER -> CLIENT
     --------------------------------------
-----------------------------------
     CONNECT home.netscape.com:443 HTTP/1.0
     User-agent: Mozilla/4.0
     <<< empty line >>>
                                             HTTP/1.0 407 Proxy auth required
                                             Proxy-agent: Netscape-Proxy/1.1
                                             Proxy-authenticate: ...
                                             <<< empty line >>>
     CONNECT home.netscape.com:443 HTTP/1.0
     User-agent: Mozilla/4.0
     Proxy-authorization: ...
     <<< empty line >>>
                                             HTTP/1.0 200 Connection
established
                                             Proxy-agent: Netscape-Proxy/1.1
                                             <<< empty line >>>
                  <<< data tunneling to both directions begins >>>
```

5. Multiple Proxy Servers

This specification applies equally to proxy servers talking to other proxy servers. As an example, double firewalls make this necessary.

In this case, the inner proxy is simply considered a client with respect to the outer proxy.

6. Security Considerations

The CONNECT tunneling mechanism is really a lower-level function than the rest of the HTTP methods, kind of an escape mechanism for saying that the proxy should not interfere with the transaction, but merely forward the data. In the case of SSL tunneling, this is because the proxy should not need to know the entire URI that is being accessed (privacy, security), only the information that it explicitly needs (hostname and port number) in order to carry out its part.

Due to this fact, the proxy cannot necessarily verify that the protocol being spoken is really what it is supposed to tunnel (SSL for example), and so the proxy configuration should explicitly limit allowed connections to well-known ports for that protocol (such as 443 for HTTPS, 563 for SNEWS, as assigned by IANA, the Internet Assigned Numbers Authority).

Ports of specific concern are such as the telnet port (port 23), SMTP port (port 25) and many UNIX specific service ports (range 512-600). Allowing such tunnelled connections to e.g. the SMTP port might enable sending of uncontrolled E-mail ("spam").

7. References

[HTTP/1.0] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol -- HTTP/1.0. RFC 1945, MIT/LCS, UC Irvine, May 1996.

[HTTP/1.1] R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext Transfer Protocol -- HTTP/1.1. RFC 2068, UC Irvine, DEC, MIT/LCS, January, 1997.

[TLS] T. Dierks, C. Allen, A. O. Freier, P. L. Karlton, and P. Kocher. The TLS (Transport Layer Security) Protocol. Internet-Draft draft-ietf-tls-protocol-05.txt, Consensus Development, Netscape Communications, November 12, 1997.

[SSL] K. Hickman, T. Elgamal, "The SSL Protocol", draft-hickman-netscape-ssl-01.txt, Netscape Communications Corporation, June 1995.

[SSL3] A. O. Freier, P. Karlton, Paul C. Kocher, "The SSL Protocol -- Version 3.0",

draft-ietf-tls-ssl-version3-00.txt, November 18, 1996.

8. Author's Address:

Ari Luotonen <ari@netscape.com>
Mail-Stop MV-068
Netscape Communications Corporation
501 East Middlefield Road
Mountain View, CA 94043
USA